Connector Subsystem

This subsystem describes the overall geometry for all connector types as well as the placement of symbols and labels on connector stems.

Relationship numbering range: R50-R99

Class Descriptions

Anchored Stem

Not to be confused with the beer made in San Francisco, California. This is a Stem whose root end is determined by a user specified Face Placement position on the Node Face.

Attributes

ID

Same as**Stem.ID**

Connector

Same as**Stem.Connector**

Node

Same as **Stem.Node**

Face

Same as **Stem.Face**

Anchor position

Relative distance from the center of the Node face.

Type: Face Placement `−5..+5` where zero represents the center with + to the right or top and - to the left or bottom, both away from the center

Identifiers

1. **ID** + **Connector**
2. **Node** + **Face** + **Anchor position**

To prevent any drawing overlap, two Stems may not anchor at the same Node face placement location.

Annotation

The application of a Label to a Decorated Stem is an Annotation. Whereas a Decoration is drawn on a Stem on one end or the other (root or vine), a Label is offset from the Stem so that it doesn't overlap the Stem line and relative to the Node face where the Stem is attached.

Attributes

Stem type

Same as **Decorated Stem.Stem type**

Semantic

Same as **Decorated Stem.Semantic**

Diagram type

Same as **Decorated Stem.Diagram type**

Notation

Same as **Decorated Stem.Notation**

Label

Same as **Label.Name**

Default stem side

By default the Rendered Label will appear on this side of the Stem axis in its vicinity. The user can override this default by specifying a Label flip. If the stem is drawn vertically near the Label, it will appear to the right or left and if the stem is drawn horizontally the label will be above or below the stem. If a + value is specified, it means to the right or above since the x or y axis increases in that direction.

Type: [+ | –]

Vertical stem offset

If the Stem is drawn horizontally, this is the vertical space between the Label content rectangle and the Stem.

Type: Distance

Horizontal stem offset

If the Stem is drawn vertically, this is the horizontal space between the Label content rectangle and the Stem.

Type: Distance

Node face offset

The minimum (and default) distance between the Label content rectangle face parallel and closest to the Node face at the root end of the Stem.

Type: Distance

Identifiers

Stem type + **Semantic** + **Diagram type** + **Notation**

Consequence of a one-many association with id formed from reference to the many side.

Connector

A Connector is a set of Stems connected by one or more lines to form a contiguous branch bringing one or more Nodes into a drawn model level relationship. On a class diagram, for example, a Connector is drawn for each binary association, generalization and association class relationship.

The Connector Type and its Stem Types determine how the Connector should be drawn.

Attributes

ID

Each Connector is numbered uniquely on its Diagram.

Type: Nominal

Diagram

Same as **Diagram.ID**

Connector type

Same as **Connector type.Name**

Diagram type

Same as **Connector type.Diagram type**

Identifiers

ID

Since only one Diagram is drawn at a time, there is only ever one instance of Diagram and so the Connector.ID suffices as a unique identifier.

Connector Layout Specification

Defines a set of values that determine how a Connector is drawn.

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 stem positions

The number of equally spaced positions relative to a center position (included in the count) on a Node face where a Stem can be attached. A value of one corresponds to a single connection point in the center of a Node face. A value of three is a central connection point with one on either side, and so on. In practice, five is usually the right number, especially for a class or state diagram. But this could vary by diagram and node type in the future.

Type: Odd Quantity **: :** Odd Integer > 0

Default rut positions

The number of ruts where Path can be defined in a Lane. These work like stem/anchor positions on a Lane as opposed to a Node face. For a value of 3 we get positions -1, 0 and +1 with 0 representing the Lane Center and +1 high/right and -1 low/left.

Type: Odd Quantity **: :** Odd Integer > 0

Default new path row height

When a new empty row must be added to to accommodate a Path in a Connector use this initial height.

Type: Distance

Runaround lane width

When a new empty row or column must be added to accommodate a Path that bends outside the grid, this is the initial height or width to use in creating that Lane.

Type: Distance

Identifiers

1. **Name** // This is a singleton, so the name is certainly unique

Connector Name

The user may supply a name for any or all Connectors in a Diagram. On a class diagram, for example, the user would specify names like R2, R35, etc. for each relationship Connector.

Attributes

Connector

Same as **Connector.ID**

Name

The user supplied name to be drawn on or near the Connector axis.

Type: Text

Bend

If the Connector is bent, we proceed clockwise from the first attached Node starting from 1 for each bend. The term "bent" can be applied liberally. In the case of a Binary Connector, we really mean Bend. With a Tree Connector, the quantity can represent each Branch.

The Name will then be placed at the center of the line segment of this Bend.

In the case of a non-bent Connector, this value is ignored

Type: Count

Side

For a horizontal Connector, this will be above or below and for a vertical Connector it will be left or right. Since both right and above are at increasing coordinate values along one coordinate axis, we can just use a positive or negative sign to indicate the Side. Positive (1) means above or right while negative (-1) is the other side.

Type: `[ 1 | -1 ]` as an integer value

Location

The coordinates of the lower left text bounding box.

Type: Position

Size

The dimensions of the text bounding box.

Type: Rect Size

Identifiers

1. **Connector type** + **Diagram type** + **Notation**

Connector Name Specification

A Diagram Notation may specify that a given Connector Type be named along with the default placement information for that name. For diagram generation purposes, we leave it to the user to supply a name with a format appropriate to the Diagram Type and Notation. But we can retain layout information so that the user need not specify precise placement of each name. For example, we can say that a certain name be placed in the center of each connector overlaying it, or at a certain distance above a horizontal connector and to the right of a vertical connector.

The name of a Connector is not associated with any particular end of the Connector. In that case you would use a Stem name instead.

Attributes

Connector type

Same as **Connector Type.Name**

Diagram type

Same as both **Diagram Notation.Diagram type** and **Connector Type.Diagram type**

Notation

Same as **Diagram Notation.Notation**

Vertical axis buffer

The buffer ensures that there is consistent whitespace between the name and the connector axis. For example, all names for a given Connector Type can be drawn with 7 points of empty space above or below the Connector line segment.

This buffer is the vertical gap above or below a horizontal connector bend. The distance is measured from the edge of the text bounding box closest to the adjacent connector axis.

If the value is zero, the name is drawn centered on top of the Connector with a solid fill around the text so that the connector line is never drawn through the text.

Type: Distance

Horizontal axis buffer

Same concept as the **Vertical axis buffer** except that this is the horizontal gap, right or left, of a vertical connector bend.

Type: Distance

Default name

A text value to be used in case the user does not supply a name.

Type: Text

Optional

Whether or not the name is required or optional. If required and no name is supplied a warning can be raised and the default name applied.

Type: Boolean

Identifiers

1. **Connector type** + **Diagram type** + **Notation**

Connector Style

Connectors are ordinarily drawn as un-patterned lines. If the lines in a Connector will be drawn with some other pattern, such as dashed, a Connector Style is defined. For example, the xUML dependency connectors in a domain diagram (package dependency) are dashed.

Attributes

Connector type

Same as Connector Type.Name

Diagram type

Same as both **Diagram Notation.Diagram type** and **Connector Type.Diagram type** This enforces the constraint that a line style can be defined only for a Notation defined on the Connector Type.

Notation

Same as **Diagram Notation.Notation**

Stroke

The stroke style to use when drawing the Connector lines.

Type: Stroke Style

Identifiers

1. **Connector type** + **Diagram type** + **Notation**

From association multiplicity

Connector Type

One or more Nodes may be interrelated by some model level relationship such as a state transition, generalization, association, dependency and so forth. Each such relationship is drawn with one or more connecting lines and terminating symbols. A Connector Type defines the symbols, line connection geometry and appearance of Connectors corresponding to some model level relationship.

Attributes

Name

The name of the model level relationship such as “Transition” or “Generalization”.

Type: Name

Diagram type

Same as **Diagram Type.Name**

Geometry

This describes the way that a Connector is drawn, pulling together all of its Stems. Many geometries are possible, but only a handful are supported which should cover a wide range of diagramming possibilities.

Unary – Relationship is rooted in some Node on one end and not connected on the other end. An initial transition on a state machine diagram is one example where the target state is connected and the other end of the transition just has a dark circle drawn at the other end (not a Node). It consists of a single Stem.

Binary – Relationship is drawn from one Node face position to another on the same or a different Node. This could be a state transition with a from and to state or a binary association from one class to another or a reflexive relationship starting and ending on the same class or state. It consists of two Stems, one attached to each Node face position connected together with a line. A Tertiary geometry where a third Stem connects a Node face to the binary connection is also possible in this geometry. It is considered an optional extension that can be defined on any Binary Connector.

Tree – Here one Node is a root connecting to two or more other Nodes. A Stem emanates from the root Node and another type of Stem emanates from each of the subsidiary Nodes and one or more lines are drawn to connect all the Stems. A class diagram generalization relationship is a typical case.

Type: Connection Geometry`:: [ unary | binary | tree ]`

Identifiers

Name + **Diagram type**

The Name is unique for each Diagram Type by policy. It seems likely that a name like “Transition, for example, could be useful and defined differently across Diagram Types.

Decorated Stem

A Stem Signification that is decorated somehow when it appears on a Diagram is considered a Decorated Stem. Not all Stem Significations are decorated. The stem attaching a class diagram subclass is not notated in many class diagram notations.

See R55 description for more details.

Attributes

Stem type

Stem Signification.Stem type

Semantic

Stem Signification.Semantic

Diagram type

Type: Same as**Stem Signification.Diagram type** and **Diagram Notation.Diagram type**

Notation

Diagram Notation.Notation

Stroke

This is the style used to draw the Stem where it isn't occluded by any Symbols. In most cases it is probably just the default connector style. But in at least the case of an `xUML- associative mult` Decorated Stem, a dashed line is typically drawn.

Type: Stroke Style

Identifiers

Stem type + **Semantic** + **Diagram type** + **Notation**

Consequence of a many-many association with a shared Diagram Type.

Floating Stem

The user specifies the Node face, but not the attachment position of a Floating Stem. The point on the Node face where a Floating Stem attaches is determined by the position of an opposing Anchored Stem so that a straight line between them is ensured.

Attributes

ID

Same as**Stem.ID**

Connector

Same as**Stem.Connector**

Identifiers

1. **ID** + **Connector**

Free Stem

This type of Stem is used to create a Unary Connector. In fact, a Free Stem comprises the entire Unary Connector.

Attributes

ID

Same as**Stem.ID**

Connector

Same as**Stem.Connector**

Identifiers

1. **ID** + **Connector**

Rendered Label

The application of a Label to a Decorated Stem is an Annotation. Whereas a Decoration is drawn on a Stem on one end or the other (root or vine), a Label is offset from the Stem so that it doesn't overlap the Stem line and relative to the Node face where the Stem is attached.

Attributes

Stem

Same as **Stem.ID**

Connector

Same as **Stem.Connector**

Location

The location of the lower left corner in Diagram coordinates

Type: Position

Stem type

Same as both **Annotation.Stem type** and **Stem.Stem type**

Semantic

Same as both **Annotation.Semantic** and **Stem.Semantic**

Diagram type

Same as both **Annotation.Diagram type** and **Stem.Diagram type**

Notation

Same as both **Annotation.Notation** and **Stem Type.Notation**

Identifiers

1. **Stem** + **Connector** / / Reference to many side
2. **Location**/ / Otherwise there could be an illegal overlap on the Diagram

Rendered Symbol

This is the Symbol as drawn on one end of a Stem on the Diagram.

Attributes

Stem

Same as **Stem.ID**

Connector

Same as **Stem.Connector**

Stem type

Same as both **Stem End Decoration.Stem type** and **Stem.Stem type**

Semantic

Same as both **Stem End Decoration.Semantic** and **Stem.Semantic**

Diagram type

Same as both **Stem End Decoration.Diagram type** and **Stem.Diagram type**

Notation

Same as both **Stem End Decoration.Notation** and **Stem Type.Notation**

End

Same as **Stem End Decoration.End**

Growth

The distance from the Stem End (vine or root) to the edge of the Symbol on the Stem.

Type: Distance

Identifiers

1. **Stem type** + **Semantic** + **Diagram type** + **Notation** + **Stem** + **Connector** + **End** / / From multiplicity

Stem

This is a line drawn from a face on a Node outward. The terminator on the Node face is the root and the terminator on the other side of the line is the vine. Both terminators are generally referred to as the Stem ends.

A Stem may be decorated on either, both or neither end. A decoration consists of a graphic symbol such as an arrow or a circle or a fixed text Label such as the UML `0..1` multiplicity text. A graphic symbol may be combined with a text Decoration such as the Shlaer-Mellor open arrow head and `c` conditionality Label combination.

Attributes

ID

Distinguishes one Stem from another within the same Connector.

Type: Nominal

Connector

Same as**Connector.ID**

Stem type

Same as **Stem Type.Name** and **Stem Signification.Stem Type**

Diagram type

Same as **Stem Type.Diagram type** and **Stem Signification.Diagram type**

Node

Same as **Node.ID**

Face

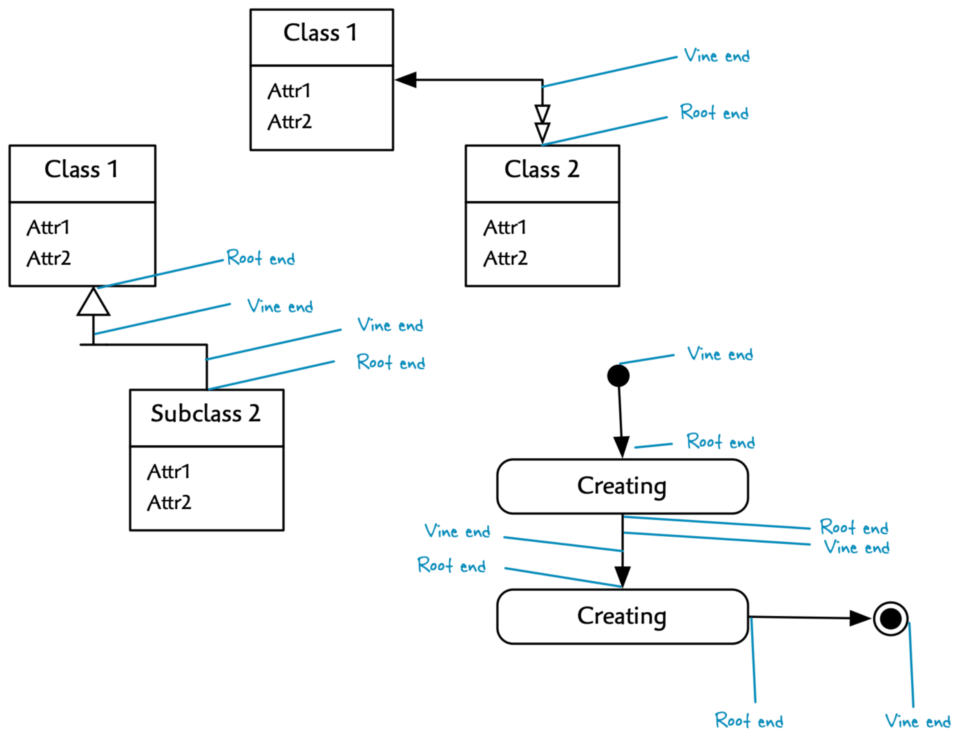
The side of the Node where the Stem is anchored.

Type: Node Face `:: [ Top | Bottom | Right | Left ]`

Root end

The point on the attached Node face where the Stem root is anchored.

Stem End positions



Type: Position

Vine end

The point where the Stem vine ends away from the attached Node. See figure in **Root end** description.

Type: Position

Identifiers

1. **ID** + **Connector**

Each Stem is uniquely numbered local to its Connector. The **ID** attribute is added since this is a -M association class which means that multiple instances of Stem may correspond to the same Connector–Stem Type pair.

2. **ID** + **Connector** + **Node** + **Face**

Superkey is provided so that Anchored Stem subclass can enforce a constraint on Stem placement to avoid coincident Stems (see Anchored Stem).

3. **Node** + **Face** + **Root end**

Now two Stems may share the same Root end position on a Node Face. Same coincident Stem constraint as supported by identifier #2 above, but enforced at the point when the coordinates are resolved.

Stem End Decoration

Either the root or vine end of a Decorated Stem that features a Symbol when drawn.

See R58 description for more details.

Attributes

Stem type

Same as **Decorated Stem.Stem type**

Semantic

Same as **Decorated Stem.Semantic**

Diagram type

Same as **Decorated Stem.Diagram type**

Notation

Same as **Decorated Stem.Notation**

Symbol

Same as **Symbol.Name**

End

A Stem has two ends, root and vine. Either, both or neither end may be decorated.

Type: `[ root | vine ]`

Identifiers

Stem type + **Semantic** + **Diagram type** + **Notation** + **Symbol**+ **End**

Consequence of a many-many association with the addition of an extra attribute **End placement** to distinguish the -M associative multiplicity.

Stem Name

The user may supply a name for any or all Connectors in a Diagram. On a class diagram, for example, the user would specify names like R2, R35, etc. for each relationship Connector.

Attributes

Stem

Same as **Stem.ID**

End

The end of the Stem where the name is placed.

Type: `[ root | vine ]`

Name

The user supplied name to be drawn on or near the Stem. The text will be right or left aligned depending on the location relative to the Stem.

Type: Text

Side

For a horizontal Stem, this will be above or below and for a vertical Stem it will be left or right. Since both right and above are at increasing coordinate values along one coordinate axis, we can just use a positive or negative sign to indicate the Side. Positive (1) means above or right while negative (-1) is the other side.

Type: `[ 1 | -1 ]` as an integer value

Location

The coordinates of the lower left text bounding box.

Type: Position

Size

The dimensions of the text bounding box.

Type: Rect Size

Identifiers

1. **Connector type** + **Diagram type** + **Notation**

Stem Name Specification

For a given Notation, certain Stem Types are named. For each case we can establish the uniform placement of such names relative to the associated Stems.

Attributes

Stem type

Same as **Stem Type.Name**

Diagram type

Same as both **Diagram Notation.Diagram type** and **Connector Type.Diagram type**

Notation

Same as **Diagram Notation.Notation**

End

The end of the Stem where the name is placed.

Type: `[ vine | root ]`

Vertical axis buffer

The buffer ensures that there is consistent whitespace between the name and the connector axis. For a Stem, this is the distance away from the Stem which should be greater than half the width of any Stem Decoration to avoid overlap.

In the vertical case, this is the vertical distance from a horizontally aligned Stem.

Type: Distance

Horizontal axis buffer

Same concept as for **Vertical axis buffer**.

In the horizontal case, this is the horizontal distance from a vertically aligned Stem.

Type: Distance

Vertical end buffer

The buffer ensures that there is consistent whitespace between the name and the root or vine end of the Stem. In the case of a root end, this is the gap between the name bounding box and a Node Face. In the case of a vine end, it depends on the connector type. For a tertiary connector, the gap is between the name bounding box and a binary connector line segment.

In general the distance should never be zero since there is a risk that the name would be drawn on top of a stem decoration. But if there is no decoration it may make sense to specify zero so that the name is drawn over the top of the stem with a solid background.

A vertical end buffer is associated with a vertical connector line segment where the name will be to the right or left of the line

Type: Distance

Horizontal end buffer

Same concept as the **Vertical end buffer** except that this is the horizontal gap, right or left, of a vertical connector line segment.

Type: Distance

Default name

A text value to be used in case the user does not supply a name.

Type: Text

Optional

Whether or not the name is required or optional. If required and no name is supplied a warning can be raised and the default name applied.

Type: Boolean

Identifiers

1. **Stem type** + **Diagram type** + **Notation** + **End**

Stem Semantic

A Stem Semantic is some notation independent meaning that can be attributed to either end (root/vine) of a Stem. When combined with a Diagram Notation, it may or may not be represented by some visual representation such as an arrow or text.

A Stem always has meaning where it attaches to its Node since the connected Node is playing some sort of role (target state, class multiplicity, subclass, etc).

Four Stem Semantics on Binary Connectors Stems

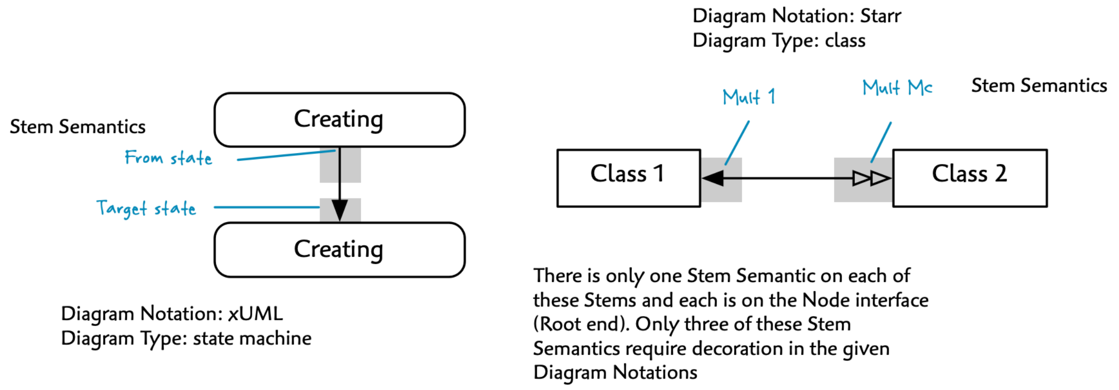
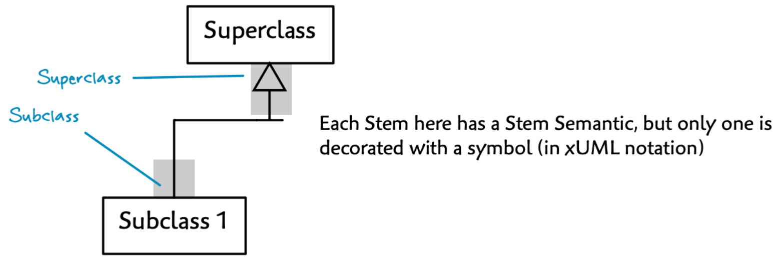


Diagram Notation: Starr
Diagram Type: class
Stem Semantics
Creating
From state
Target state
Creating
Diagram Notation: xUML
Diagram Type: state machine
Mult 1
Mult Mc
Stem Semantics
Class 1
Class 2
There is only one Stem Semantic on each of these Stems and each is on the Node interface (Root end). Only three of these Stem Semantics require decoration in the given Diagram Notations

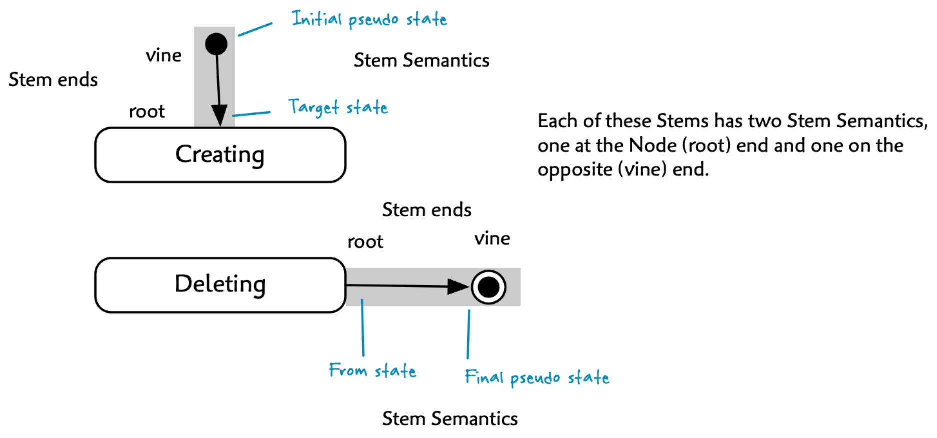
In a given Diagram Notation, a Stem Semantic may or may not require any Symbols or Labels. The from state semantic, for example, is just an undecorated line in xUML. Subclasses in xUML, Starr and Shlaer-Mellor class diagrams are similarly undecorated.

In a given Notation, not all Stem Semantics require symbolic representation.


Superclass
Superclass
Subclass
Subclass 1
Each Stem here has a Stem Semantic, but only one is decorated with a symbol (in xUML notation)

In some cases, the Stem end away from the Node face (vine end) will also have significance. Usually this is only the case when the Stem is not connected to any other Stem as it is in a Unary Connector.

Two Stems on Unary Connectors



On a state machine diagram, for example, the line that touches a state Node (root end) is terminated with an arrow to indicate a target state. The opposing end of the Stem (vine end) is undecorated unless the state Node is an initial state. In this case there is a decoration on each end of the Stem.

In the case of a deletion transition on a state machine diagram, the root end of the Stem attached to the Node is undecorated while its opposite vine end features a dot filled circle.

Attributes

Name

A name that reflects the meaning (semantic) of the Stem termination such as “target state” (goes to this state) or “Mc mult” (many conditional multiplicity) or “final psuedo-state”. Care is taken to describe meaning and not notation.

Type: Name

Identifiers

Name

Unique by policy

Stem Signification

This is a meaning that is relevant to a particular Stem Type. See the description of R62 for more details.

Attributes

Stem type

Type: Same as **Stem Type.Name**

Semantic

Type: Same as **Stem Semantic.Name**

Diagram type

Type: Same as both **Stem Semantic.Diagram type** and **Stem Type.Diagram type**. It establishes the constraint that a Stem Type may signify only a Stem Semantic that is defined on the same Diagram Type.

Identifiers

1. **Stem type** + **Semantic** + **Diagram type**

Determined by the association multiplicity

Stem Type

Defines the characteristics of the portion of a Connector attached to a Node called a 'Stem'.

In a binary association connector of a class model, for example, there are two `class mult` Stem Types and one `associative mult` Stem Type defined. A transition Connector Type in a state machine diagram defines two Stem Types, `to state` and `from state`.

Characteristics of primary interest are the semantics and notation and any other visual aspects of a Stem.

Attributes

Name

Describes the type of Node to which a Stem will be attached such as to state or association class.

Type: Name

Diagram type

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

Connector type

Type: Same as **Connector Type.Name**

About

A description of the purpose and usage of this Stem Type

Type: Description

Minimum length

A Stem of this type can never be shorter than this length. This keeps a bend or the Diagram edge from getting too close to the Node face. You wouldn't want to bend at 90 degrees less than a point away from a Node face, for example.

This value also serves to provide a default distance between the Root and Vine Ends, thus readily establishing the coordinate of the Vine End (assuming the Stem's Vine end isn't based on some other factor. In the case of a Tertiary Stem in a Binary Connector, for example, the Vine End will extend out to the nearest normal connector line, thus exceeding the Minimum Length usually.

Type: Distance

Identifiers

1. **Name** + **Diagram type**

Stem Type Names are unique to each Diagram Type by policy

Unary Connector

This type of Connector is rooted on some Node face with a vine end that does not attach to anything. It is therefore placed at some fixed distance away from the root end. The initial and final psuedo-transitions on a UML state machine diagram are both examples of Unary Connectors.

Attributes

ID

Same as **Connector.ID**

Identifiers

ID

Relationship Descriptions

R50 / 1:Mc

Connector Type can be drawn in *exactly one* **Diagram Type**

Diagram Type can draw *zero, one or many* **Connector Type**

These are the types of Connectors that can be drawn on a given Diagram Type. On an xUML state machine diagram you can draw initial, final and normal transitions, for example, whereas on an xUML class diagram you can draw generalizations, binary associations and association class relationships. More to the point, you cannot draw a state transition on a class diagram. So this relationship constrains what can be drawn on a given Diagram Type. (Though nothing prevents you from defining a new Diagram Type where this would be possible!)

Most Diagram Types will have at least one kind of Connector Type, otherwise the associated diagrams will just be a layout of unconnected Nodes. That said, there is no reason to require connections on any given Diagram Type.

A Connector Type is defined exclusively to a Diagram Type. Thus, transition on a state machine diagram may be defined differently than transition on some other kind of diagram.

Formalization

Reference in the Connector Type class

R51 / 1:Mc

Connector Type specifies *zero, one or many* **Connector**

Connector is specified by *exactly one* **Connector Type**

This is a standard specification relationship where the Connector Type defines various characteristics of a Connector. Whereas a Connector Type defines properties of all Connectors, a Connector is a manifestation of a Connector Type actually drawn on a Diagram.

When a Connector is created, it will need to grow a Stem for each connected Node and then draw a line that ties the Stems all together.

Formalization

Reference in the Connector class

R52 / 1:Mc

Node is source of *zero, one or many* **Stem**

Stem is rooted in *exactly one* **Node**

The root end of Stem is always attached to a single Node. In fact, a Stem never attaches more than one Node, though a Connector certainly can via multiple Stems. There is no such thing as a free floating Stem unattached to any Node.

A Node, on the other hand, may or may not be part of a connection. A free floating unconnected Node will not be attached to any Stem.

Formalization

Referential attribute in Stem class

R53 / M:Mc-M

Connector sprouts as *one or many* **Stem Type**

Stem Type sprouts in *zero, one or many* **Connector**

A Connector is drawn by creating all necessary Stems and then connecting them together with one or more lines. The **Connector Type.Geometry** attribute determines how these Stems and connecting lines will be drawn.

The same Stem Type may be used multiple times in a Connector. For example, an xUML class diagram binary association will need two class multiplicity Stems, one for each side of the Connector. A class diagram generalization will need one subclass stem for each subclass Node. Each connection to a Node will result in a new Stem.

If no Connectors have been drawn that use a particular Stem Type, that Stem Type will just be a definition that hasn't been used yet. In this case the Stem Type won't refer to any Connectors.

Formalization

Stem association class

R54 / 1c:Mc-1

Decorated Stem is annotated by *zero or one* **Label**

Label annotates *zero, one or many* **Decorated Stem**

A Decorated Stem may or may not have an associated text Label. In the Starr class diagram notation a generalization arrow has no associated text. In xUML, however, the arrow is accompanied by the UML tag `{ disjoint, complete }`. There seems to be no reason to support multiple fixed text Labels as none of the supported notations require them.

A given Label may be used with more than one Decorated Stem. The Shlaer-Mellor `c` label is associated with any class multiplicity where zero is a possibility, for example.

A Label may be defined that is not used with any notation, though this is unlikely. It can be done in anticipation of supporting a future notation, however.

Formalization

Referential attributes in the Annotation class

R55 / Mc:Mc-1

Diagram Notation decorates *zero, one or many* **Stem Signification**

Stem Signification is decorated with *zero, one or many* **Diagram Notation**

Each Diagram Notation may specify a different decoration for a Stem Signification. The Starr class diagram notation, for example assigns a double hollow arrow at the root end of a `class mult – Mc mult` Stem Signification. xUML, on the other hand specifies only a text label of `0..*`for that same Stem Signification.

In fact, a Stem Signification may not be decorated at all in a given Diagram Notation. The `from state – source state` Stem Signification on a state machine diagram, for example, is not decorated in xUML while the `to state – target state` is.

A given Diagram Notation only specifies decoration for those Stem Significations relevant to the associated Diagram Type. Thus the, `Starr – class` Diagram Notation does not specify decoration on any Stem Significations on a state machine diagram.

Formalization

Stem Decoration association class

R56 / 1:Mc

Stem indicates *one* **Stem Signification**

Stem Signification is indicated on *zero, one or many* **Stem**

When a Stem is drawn it binds to one of the Stem Significations that its Stem Type may signify. A Stem whose type is `class mult` (class multiplicity) must indicate one of the available multiplicity significations, namely: `1`, `M`, `1c` or `Mc`. The selection will be user specified. For many Stem Types there will be only one Stem Signification to choose from so the indication is automatic.

Formalization

Referential attributes in the Stem class

R57 / 1:Mc

Diagram Type is context for *zero, one or many* **Stem Semantic**

Stem Semantic has meaning on *exactly one* **Diagram**

Consider a Stem Semantic such as `class mult` (class multiplicity) or maybe another `target state`. Each Stem Semantic defines the meaning associated with the point where a Connector attaches to some Node. The `class mult` Stem Semantic only makes sense on a class diagram while the `target state` Stem Semantic is intended for state machine diagrams.

In fact, each Stem Semantic is specific to the context defined by a type of Diagram. In other words, each Diagram Type establishes a set of relevant Stem Semantics that make sense only on that Diagram Type.

True, you may create a Diagram Type with semantics similar or almost identical to another Diagram Type. Say you define a `petri net` Diagram Type which also specifies `target state`. We still want to keep the semantics custom specified for each Diagram Type so that we don't elide subtle distinctions among them. No problem since the name of a Stem Semantic is local to its own Diagram Type. Thus a `petri net-target state` is distinct from a `state machine-target state`. The semantics may be equivalent or slightly different, but they are two distinct semantics as far as Flatland is concerned.

If a Diagram Type does not specify any Stem Semantics, this means that the Diagram Type does not support Connectors of any type. Perfectly legal, but of questionable value. Flatland will draw them at any rate!

Formalization

Reference in Stem Semantic class

R58 / Mc:Mc-M

Decorated Stem is terminated by *zero, one or many* **Symbol**

Symbol terminates *zero, one or many* **Decorated Stem End**

Each end of a Decorated Stem may or may not be adorned by a single Symbol. Keep in mind that a Symbol can be compound and built up from many graphical elements. So each terminal can be as ornate as necessary. This effectively means that at most two Symbols can be associated with a given Decorated Stem. See note in formalization section below to see how the two-ness constraint is addressed.

It is also possible for the same Symbol to be used at both ends of a Decorated Stem. Consequently this relationship is many-associative. (A given pairing of Decorated Stem and Symbol can result in two association class instances, differentiated by the **End** component of the class identifier).

If neither end of a Decorated Stem features a Symbol, there may be a Label associated with the Stem. If there is no Label either, perhaps the Stem is notated by changing its line stroke pattern. For example, in xUML an associative 1 multiplicity on a class diagram is shown by drawing the stem as a dashed pattern with no other label or symbol.

A Decorated Stem that does not have a special line pattern, Symbol or Label is not decorated and should not be declared as such. No harm can come from falsely declaring a Decorated Stem with no Decoration, it will just be rendered as a linear Stem, but it is bad practice.

A given Symbol can be used in as many Decorated Stems as you like. A `solid arrow` for example might be used both in a state transition and in a domain diagram dependency. If a Symbol is not used at all, there is no harm as it may become useful in a Diagram Notation defined later.

Formalization

Referential attributes in the Stem End Decoration association class along with enforcement of the two-ness constraint. This is accomplished by integrating the **End** attribute into the Decoration identifier. See the class description for more details.

R59 / 1:M

Connector Type connects nodes with *one or many* **Stem Type**

Stem Type defines node connections for *one or many* **Connector Type**

We define the structure of a Connector by describing it as a set of Stems of various types that are lashed together with connecting lines. Each Stem Type establishes the meaning of the interface between a Connector line and the Node where it attaches. For each type of Connector, certain types of Stems are relevant.

For example, a generalization Connector Type defined on a class diagram requires only two types of Stems, a superclass and a subclass Stem Type to designate the meaning of each connection point. Furthermore, the subclass Stem Type has relevance only to a class diagram generalization Connector Type.

A Connector Type without any Stem Types makes no sense because it couldn't connect to any Nodes. And a Stem Type only has utility as part of some Connector Type.

Formalization

Stem Type Usage association class with shared **Diagram type**

R60 / Mc:Mc-1

Connector Type lines are styled in *zero, one or many* **Diagram Notation**

Diagram Notation styles lines of *zero, one or many* **Connector Type**

A Connector Style is defined only for those Connector Types that are not simple solid black lines. So most Connector Types do not need any special style in a given Diagram Notation.

A given Diagram Notation may or may not set styles for Connector Types.

Formalization

References in the Connector Style association class

R61 / Mc:Mc-1

Stem End Decoration is rendered near *zero, one or many* **Stem**

Stem renders *zero, one or many* **Stem End Decoration**

When a Stem is drawn, any corresponding Symbols are positioned on one or both Stem end axes and rendered as specified by the Stem End Decoration. (There are at most two Symbols placed on a given Stem, one at each end).

Formalization

Referential attributes in the Rendered Symbol class

R62 / M:M-1

Stem Type may signify *one or many* **Stem Semantic**

Stem Semantic may be signified by *one or many* **Stem Type**

A Stem Semantic refines the general meaning specified by a Stem Type. A `class mult` Stem Type, for example, indicates the dual concepts of multiplicity and conditionality. A variety of Stem Semantics are available that each establish a precise pairing of multiplicity and conditionality `1 mult` (unconditional 1), `Mc mult` (conditional many), and so forth. When a Stem is created, it must bind to one of the Stem Semantics available to the Stem's Stem Type.

A given Stem Semantic may be relevant to more than one Stem Type. The unconditional multiplicity `1 mult` and `M mult` Stem Semantics, for example, also apply to the `associative mult` Stem Type that defines the Stem on a class diagram's association class.

A Stem Semantic is not useful if it has no relevance to any Stem Type, so it must be relevant to at least one.

Many Stem Types have meanings that cannot be further modified and therefore may signify only one available Stem Semantic. A `to state` Stem Type can only mean `target state`, for example. But every Stem Type does not have a specific meaning unless it can signify at least one Stem Semantic.

Formalization

References in Stem Semantic Option association class

R63 / 1:Mc

Diagram shows *zero, one or many* **Connector**

Connector appears on *one* **Diagram**

A Connector is rendered on the one and only Diagram. And it is certainly possible to create a Diagram with Nodes and no Connectors.

Formalization

Reference in Connector class

R65 / Generalization

Anchored Stem is an **Anchored Binary Stem**, **Tertiary Stem**, **Anchored Tree Stem** or **Free Stem**

Each of these subclasses of Anchored Stem are Stems that attached at a user specified anchor position on a Node face.

Each Connector subclass determines the quantity and combination of various types of Stems. A Tree Connector, for example, consists of one Trunk Stem and one or more Leaf Stems. A Unary Connector consists of a single Free Stem.

See each relevant connector subsystem to see how each subclass of Anchored Stems are applied.

Formalization

ID + Connector referenced from each subclass

R66 / Generalization

Floating Stem is a **Floating Binary Stem** or **Floating Leaf Stem**

Floating Stems have utility in both the Binary and Tree Connector subsystems. Though they play different roles in each, a Floating Stem always derives its axis from a coincident Anchored Stem guide.

Formalization

References in the subclasses

R67 / Generalization

Stem is a **Floating Stem** or **Anchored Stem**

An Anchored Stem is positioned by the user with an **Anchor position**. This position is later resolved to diagram coordinates. Anchored Stems are used in all Connector Types.

A Floating Stem is lined up with an opposing Anchored Stem so that a straight line is formed. The pairing of Anchored and Floating Stems is useful in both Binary and Tree Connectors.

With a Straight Binary Connector, there is no need for two user specified anchor positions. Since the Connector is a straight line, only one anchor position is necessary. In fact, there should only be one to ensure that we end up with a non-diagonal line when the coordinates are resolved.

The non-anchored Stem in a Straight Binary Connector is understood to float so that it is level with the opposing Anchored Stem. The position of a Floating Binary Stem is computed for a horizontal line by sharing the x coordinate of the opposing Anchored Stem. This is the y coordinate if the line is vertical.

The same situation can occur in a Tree Connector where one Leaf Stem is anchored while another is lined up with it straight across.

Formalization

ID + Connector in either subclass or **ID + Connector + Stem type + Node + Face + Anchor position** in the Anchored Stem subclass. Two different ID's are referenced since the Anchored Stem is enforcing a constraint preventing two Anchored Stems from being placed in the same location on the same Node face.

R68 / 1c:Mc-1

Annotation is rendered near *zero, one or many* **Stem**

Stem renders *zero or one* **Annotatio**

When a Stem is drawn, any corresponding Label is positioned on the Diagram and rendered as specified by the Annotation.

Formalization

Referential attributes in the Rendered Label class

R69 / Generalization

Connector is a **Hierarchy**, **Unary** or **Binary Connector**

Different rules and constraints may apply to each geometry so they are subclassed. Primarily an unbent Binary Connector has a special relationship to a Floating Stem.

The type is determined by the **Connector Type.Geometry** attribute where both binary and tertiary geometries are folded into the Binary Connector and distinguished by the **Binary Connector.Tertiary** stem boolean attribute.

Formalization

The identifier in each of the subclasses referring to the superclass identifier