

CHAPTER 6

Diagramming and Model Organization

Diagramming involves the use of diagrams to pictorially or graphically render a model.

Model organization involves the use of packages to organize the various elements of a model.

Diagrams

Diagrams render views of a model pictorially or graphically. Diagrams

- Are drawn at a level of abstraction or detail and from a viewpoint or perspective.
- Are graphical representations of a set of model elements. These model elements include classes, objects, associations, and links.
- Contain graphical constructs or symbols, which
  - Are model elements, relationships among model elements, and information associated with model elements.
  - May be nodes (vertices), paths (arcs), or strings.
  - May be suppressed on a diagram when they are not relevant or of interest.
  - May have adornments or attached graphical constructs that provide information.
- Contain visual relationships among graphical symbols, which
  - Are connection relationships, where paths are drawn to nodes.
  - Are containment relationships, where graphical constructs are within nodes.
  - Are attachment relationships, where strings are near paths or nodes.

- May be defined as required. There are nine predefined types of diagrams: class, object, use case, sequence, collaboration, statechart, activity, component, and deployment diagrams. New types of diagrams (models) may be defined as required.
- May have stereotypes and properties that apply to every element in the diagram (Chapter 15).

Nodes

Nodes (Figure 6-1) are graphical constructs that render model elements.

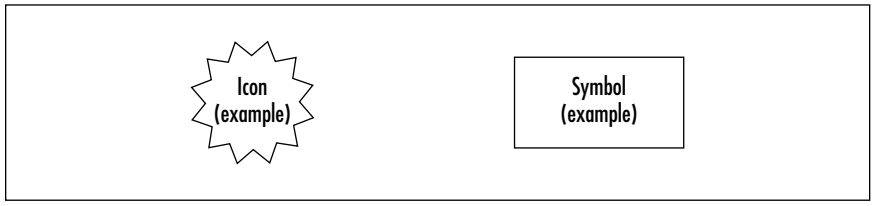


Figure 6-1: Nodes

Nodes

- May be icons, that is, graphical figures of fixed size and shape that do not expand to hold content.
- May be two-dimensional symbols, that is, figures of variable height and width that expand to hold content within their boundaries. These symbols may expand to hold lists of strings or other symbols, and they may also be divided into compartments in which to hold their contents.
- May reside in containers.
- May have names and associated information contained in, below, or above their symbols or icons.
- May have stereotypes and properties (Chapter 15).

Paths

Paths (Figure 6-2) are graphical constructs that render model element relationships between other model elements.

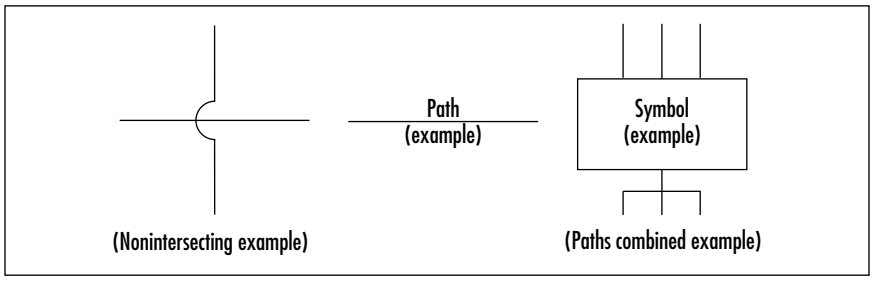


Figure 6-2: Paths

Paths

- Are sequences of line segments whose endpoints connect or attach to nodes.
- May consist of orthogonal, oblique or diagonal, and curved line segments.
- May cross one another without intersecting, as depicted by a small semi-circular jog by one of the paths.
- May be part of a set of paths that individually connect to a single node. These paths may be combined into a single path that branches into several paths in a treelike manner provided that the original paths of the set are of the same kind and that the modeling information on those paths is identical.
- May reside in containers.
- May have names and associated information contained below or above their line segments.
- May have stereotypes and properties.

Strings

Strings are sequences of text characters. Strings

- Are used to present information in an unparsed form.
- May present underlying model information.
- May exist as standalone elements on diagrams.
- May be a single line or a paragraph.
- May exist as singular elements of symbols or within compartments of symbols.
- May exist as elements in lists or as labels attached to symbols or paths. The position of a string in a list may also convey information.
- May express information in a particular language (pseudocode, computer language, natural language, or the OCL). This applies to the various types of strings used in diagrams.

Names

Names (Figure 6-3) are strings that are attached to other graphical constructs.

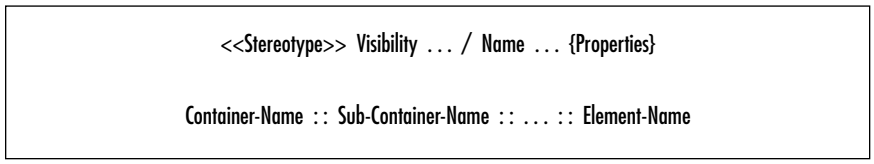


Figure 6-3: Names

Names

- Are also known as selectors or qualifiers.
- Are used to uniquely identify model elements within some container or context. They must be unique within their container or the context in which they are used. Furthermore, OCL expressions may be used to resolve ambiguities

in referencing elements within a given context or among a collection of related model elements (Chapter 16).

- May have a stereotype keyword enclosed in guillemets (<< >>), or double angle brackets, preceding the name (Chapter 15).
- May have properties enclosed in curly braces following the name (Chapter 15).
- May have a visibility symbol or character indicating the visibility of an element outside of its container. The visibility symbol is specified for an element when it is declared on a structural diagram.
  - "+" indicates public visibility. Elements are available to importing containers (packages using the import dependency) and specialized or subtype containers. Public elements within classes are accessible from outside their container.
  - "#" indicates protected visibility. Elements are available to specialized or subtype containers. Protected elements within classes are accessible from descendants of the container.
  - "-" indicates private visibility. Elements are not available outside their container. Private elements within classes are not accessible from outside their container.
  - The visibility of an element when exported to an importing container is the most restrictive of the element's visibility and its container's visibility. Elements of imported containers may be referenced by pathname or an alias if visibility permits. Elements of inherited containers may be referenced by name if visibility permits.

  The absence of a visibility marker indicates that visibility is not shown. This does not mean that it is undefined or has a default value.
- May have other information around the name. This is determined by the particular use of the name and the type of modeling element to which it is attached.
- May be alias names. When a container receives an element from another container, the received element may have the same name as an already existing element in the receiving container. To avoid a name conflict, where two elements have the same name, the received element is given a pseudo or alias name. The original name is attributed to the element originally in the container, and the alias name is attributed to the element received from another container.
- May be pathnames that identify model elements from some starting point, such as the root of the system. This is a sequence of names concatenated by a delimiter (two colons). Each name references an inner container, and the last name in the sequence references the actual element.
  - When a pathname is used to reference an element, the element is defined elsewhere.
  - When an element is explicitly named, the element is defined within the context.
- May be attached to derived elements, that is, elements that can be computed from other elements. A slash (/) in front of the name is used to specify a derived element. The details of computing the elements may be attached in a constraint string (Chapter 15).

Labels

Labels are strings that are attached to other graphical constructs. They are used to represent information or adornments attached to graphical constructs on a diagram.

Expressions

Expressions are strings consisting of linguistic formulas that yield values of a particular type when evaluated. Expressions

- Must be expressed in a particular language (pseudocode, computer language, natural language, or the OCL). This applies to the various types of expressions used in diagrams.
- May be run through a linguistic analyzer to yield values of the appropriate type and express their meaning.
- May yield results of a given type. The OCL specifies a set of predefined types (Chapter 16).

Keywords

Keywords are text strings that represent a variation on a common theme. Keywords

- Must be enclosed in guillemets (<< >>). Double greater-than and double less-than characters may be used.
- That are predefined are treated as reserved words and may not be redefined by a user.
- May be added by users as stereotype names.

Compartments

Compartments (Figure 6-4) are divisions of two-dimensional symbols.

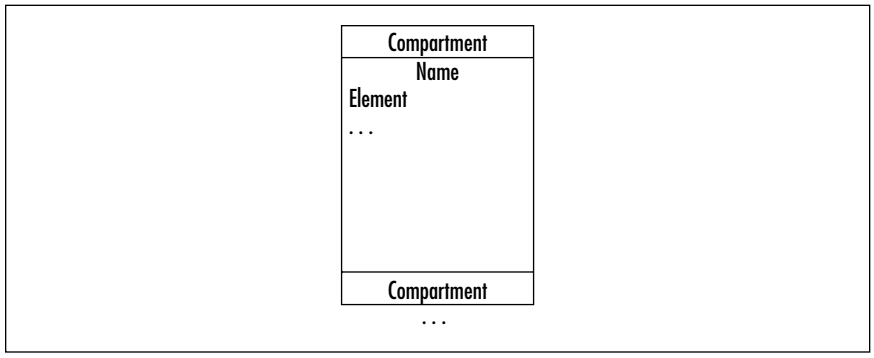


Figure 6-4: Compartments

Compartments

- Are used to partition a two-dimensional symbol so that specific semantics may be associated with a compartment.

- Are indicated by dividing a symbol using horizontal separator lines.
- May be suppressed, in which case no separator line is shown.
- May have a compartment name. Compartment names are typically centered, in plain type, and located at the top of the compartment.
- May contain a list of names or strings denoting elements of the compartment.
- May contain a subdiagram of the elements within the compartment.

Lists

Lists (Figure 6-5) are a sequence of names or strings.

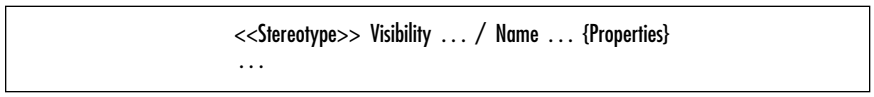


Figure 6-5: Lists

Lists

- Are rendered one element per line.
- May be contained within compartments.
- May contain stereotype keywords. Such keywords apply to all succeeding elements of the list until another stereotype keyword list element appears, an empty stereotype keyword nullifies it, or the end of the list is reached.
- May contain property strings, which apply to all succeeding elements of the list until another property string list element appears or the end of the list is reached. Property strings applied to a list of elements are not superseded by property strings attached to individual elements, but may be augmented or modified by them.
- May have an ellipsis (...) as the final element to indicate that there exist additional elements that are not shown.
- May be sorted or filtered according to some selection rule.

Multiplicity

Multiplicity (Figure 6-6) is a string that specifies a range of allowable cardinalities for a set. Cardinality is explicitly the number of elements in a set, which can be an infinite subset of the nonnegative integers.

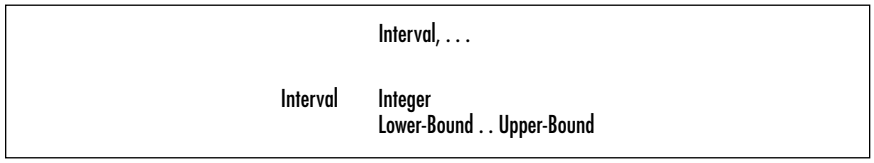


Figure 6-6: Multiplicity

Multiplicity

- Is a comma-separated sequence of integer intervals.

- May have literal integer values.
- May have closed ranges of integers. Ranges are inclusive; that is, they include their boundaries.
- May use the asterisk character (*) for the lower bound or upper bound to denote an unlimited bound.
- May use a single asterisk (*) to denote the unlimited nonnegative integer range; it is equivalent to *..* and 0..* (zero or more).
- Must not be 0..0 since this would not allow an element to exist at all.
- Must have monotonically increasing (or decreasing) intervals; that is, it must be ordered by increasing (or decreasing) intervals.
- Must have contiguous intervals combined into a single interval; that is, when one interval ends where the next interval begins, they are combined into one interval.
- May explicitly show the lower bound and upper bound. That is, "1" may be shown as "1..1," and "*" shown as "0..*". This notation explicitly indicates that both bounds have been considered.

Type-Instance Correspondence

A type-instance dichotomy (Figure 6-7) allows a generic descriptor to describe many individual items (nodes, paths). This correspondence between a generic descriptor and its individual items adheres to the following properties:

- The generic descriptor is known as a typelike element and is a class of elements.
- The individual items are known as instancelike elements and are objects of a class of elements.
- Generic descriptors
  - Are rendered using a geometric symbol.
  - Must have a name or identifier string that represents the name of the generic descriptor.
  - Contain a description of the content of the descriptor.
- Individual items
  - Use the same geometric symbol employed by their generic descriptor.
  - May have a name or identifier string that represents the name of the item. The name must be underlined. When omitted, the colon remains and the item is considered an anonymous item.
  - May have a colon followed by a name or identifier string that represents the name of the item's generic descriptor. The name must be underlined.
  - Contain values for their contents
- The stereotype of an instancelike element must match the stereotype of its typelike element (Chapter 15).

Notes

Notes (Figure 6-8) are graphical constructs or notational items containing and rendering textual information.

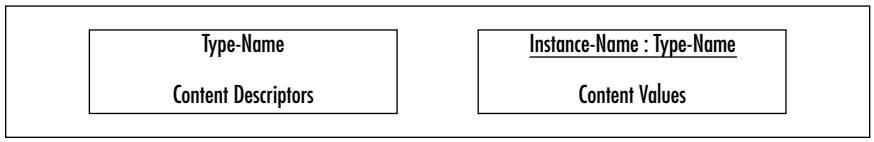


Figure 6-7: Type-Instance Correspondence

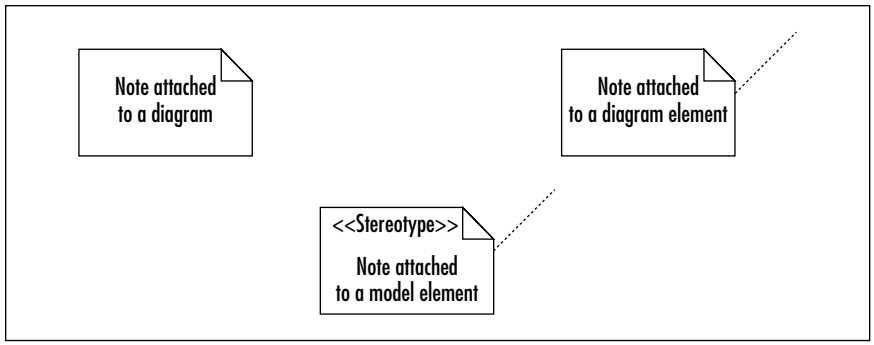


Figure 6-8: Notes

Notes

- Are denoted as rectangles with a bent upper right corner.
- Are used to render comments, constraints, properties, method bodies, and so forth.
- May be attached to zero or more model elements or graphical constructs by dashed lines.
- May be comments that do not have any underlying model elements and are attached to the diagram itself or to a diagram element.
- May be stereotyped to designate that it is part of the model and not just part of the diagram or view.

Packages

Packages (Figure 6-9) are general-purpose grouping mechanisms used to organize semantically related model elements.

Packages

- Are denoted as large rectangles with a small rectangle attached on one corner (usually the left upper side of the large rectangle).
- Are used to define a namespace or context (container) wherein each name must be unique.
- May contain subordinate or nested packages and ordinary model elements.
- May be used at the highest level to denote a system.

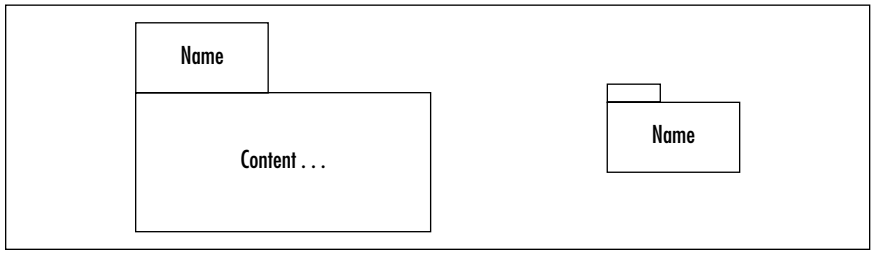


Figure 6-9: Packages

- Must not be instantiated. Packages only serve as containers for elements that may be instantiated; however, packages are not instantiated.
- Own their contents. Each element may only be owned by a single package, so that an ownership hierarchy or tree results.
- May have their contents visually suppressed.
- May contain elements that have relationships within the same package.
- May contain elements that use other elements from other packages, so that a usage network results.
- May have relationships to other packages, indicating that the relationships exist between at least some elements they contain. Such relationships are transitive; that is, if package *A* imports package *B,* and package *B* imports package *C,* package *A* also imports package *C.*
- Must use element pathnames to reference elements outside the current scope or context.
- May be nested. This establishes an implied import relationship. That is, each nested package imports the contents of its parent package.
- May have an import dependency (association) to other packages. This association must use the "import" stereotype keyword. The importing package may reference all public elements of the imported package. This represents a has-a relationship in which the importing package has the elements of the imported package.
- May have generalization relationships to other packages. The more specific package may reference all public and protected elements of the more general package. This represents an is-a-kind-of relationship in which the subpackage is a kind of superpackage.
- May be stereotyped using the "system" keyword (Figure 6-10). System packages
  - Contain the entire set of models for a complete system; this includes a specification model containing specification elements and a realization model containing realization elements.
  - Must have the requirements specified by the specification model fulfilled by the realization model.

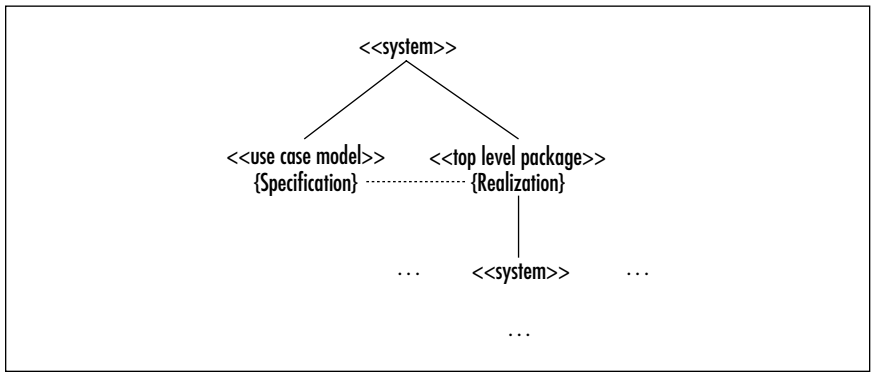


Figure 6-10: Systems, Use Case Models, Top-Level Packages, and Subsystems

- Must contain a specification model consisting of a package stereotyped using the "use case model" keyword. This package contains specification elements; they specify the behavior of the containing system and establish the boundary of the containing system. This package consists of use cases (Chapter 8) that define the operations and interfaces of the system. The specification model is at a high level of abstraction relative to the realization model.
- Must contain a realization model consisting of a set of packages. The topmost package is stereotyped using the "top level package" keyword. Other subordinate packages are stereotyped using the "system" keyword. Subsystem packages contain realization elements; they realize or offer the behavior of the containing system. Instances of realization elements collaborate to perform the specified behavior of use cases or operations and interfaces within the containing system. Because each subsystem package is a system, it may have its own specification and realization models and elements.
- May be used as classes. That is, they may be instantiated and may have associations. The package itself is not literally instantiated; rather, its contents are instantiated.

The Role of Tools

Diagramming tools play a key role in creating and maintaining diagrams. When working with diagramming tools, the following aspects should be considered:

- Diagrams on a piece of paper are static and do not have any invisible hyperlinks.
- Diagrams on a computer screen viewed using a tool are dynamic and may contain additional invisible hyperlinks that can be invoked dynamically to access other information.
- Each graphical construct, including a diagram itself, may have its own presentation style (style sheet).

- Information may also be presented in textual or tabular format rather than only graphical format.
- Tools should maintain consistency across diagram types.

,ch06.17884 Page 140 Wednesday, May 9, 2001 4:03 PM