

Glossary Of Terms

abstract class

A class that cannot be directly instantiated. No objects of this class may exist, but objects of concrete subclasses may exist. Contrast: *concrete class*.

abstraction

The essential characteristics of an entity that distinguish it from all other kind of entities. An abstraction defines a boundary relative to the perspective of the viewer.

action

A piece of behaviour performed by an object. May be implemented as an operation or as part of an operation.

action expression

A collection of actions.

activity diagram

A special case of a state diagram in which the states represent objects performing behaviour, and in which all or most of the transitions are automatic, i.e. triggered by the completion of a state.

Contrast: *state diagram*.

actor

A user of Use Cases, or the receiver of Use Case output, or both. Actors are defined to be outside of the control of the system.

aggregate

A class that represents the 'whole' in an aggregation (whole-part) relationship.

aggregation

A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part. Contrast: *composition*.

analysis

The part of the software development process whose primary purpose is to formulate a model of the problem domain. Analysis focuses what to do, design focuses on how to do it. Contrast: *design*.

architecture

The organizational structure of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts.

argument

A specific value corresponding to a parameter.

artefact

A piece of information that is used or produced by a software-development process. An artefact can be a model, a description or software.

association

The relationship between two or more classes whereby objects of one class may be linked to objects of the other class. The links are instances of the association.

association class

A class attached to an association such that for each instance of the association one object of the association class exists. The association class is a real class and may have its own associations, attributes, etc.

association end

The endpoint of an association, which connects the association to a class.

asynchronous action

A request where the sending object does not pause to wait for results. Contrast: *synchronous action*.

attribute

A part of a class that specifies which values can be held by instances of the class.

behaviour

The set of operations provided by a class.

binary association

An association between two classes. A special case of an n-ary association.

binding

The creation of a real class from a parameterised class and argument(s).

boolean

An enumeration whose values are true and false.

boolean expression

An expression that evaluates to a boolean value.

class

A description of a set of objects that share the same attributes, operations, methods, relationships and semantics.

class diagram

A diagram that shows a collection of declarative (static) model elements, such as classes, their contents and relationships.

collaboration diagram

The pre-UML2 name for the communication diagram. See: *communication diagram*.

communication diagram

A diagram that shows interactions organised around instances and their links to each other. Unlike a sequence diagram, a communication diagram highlights the relationships among the instances. Sequence diagrams and communication diagrams express similar information, but show it in different ways. Renamed from *collaboration diagram* in UML2. See: *sequence diagram*.

component

A physical entity such as a source or binary file, or an executable module.

component diagram

A diagram that shows the organisations and dependencies among components.

composite state

A state that consists of either concurrent substates or disjoint substates.

composition

A form of aggregation with strong ownership and coincident lifetime as part of the whole. Parts with non-fixed multiplicity may be created after the composite itself, but once created, they live and die with it (i.e. they share lifetimes). Such parts can also be explicitly removed before the death of the composite. Composition may be recursive.

concrete class

A class that can be directly instantiated. Contrast: *abstract class*.

concurrency

The occurrence of two or more activities during the same time interval. Concurrency can be achieved by interleaving or simultaneously executing two or more threads. See: *thread*.

concurrent substate

A substate that can be held simultaneously with other substates contained in the same composite state.

constraint

A semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML. See: *tagged value*, *stereotype*.

container

An instance that exists to contain other instances, and that provides operations to access or iterate over its contents. For example, arrays, lists and sets.

datatype

A type whose values have no identity. Datatypes include primitive built-in types (such as numbers and strings) as well as enumeration types (such as boolean).

delegation

The ability of an object to issue a message to another object in response to a message.

dependency

A relationship between two modelling elements, in which a change to one modelling element (the independent element) will affect the other modelling element (the dependent element).

deployment diagram

A diagram that shows the configuration of run-time processing nodes and the components, processes and objects that live on them. Components represent run-time manifestations of code units. See: *component diagrams*.

derived element

A model element that can be computed from another element, but that is shown for clarity or that is included for design purposes, even though it adds no semantic information.

design

The part of the software development process whose primary purpose is to decide how the system will be implemented. During design, strategic and tactical decisions are made to meet the required functional and quality requirements of a system.

device

A stereotype of actor. Used for systems that are mainly hardware rather than software.

development process

A set of partially-ordered steps performed for a given purpose during software development, such as constructing models or implementing models.

diagram

A graphical presentation of a collection of model elements, most often rendered as a connected graph of arcs (relationships) and vertices (other model elements). UML supports the following diagrams: class diagram, object diagram, use case diagram, sequence diagram, collaboration diagram, state diagram, activity diagram, component diagram and deployment diagram.

disjoint substate

A substate that cannot be held simultaneously with other substates that are contained in the same composite state.

domain

An area of knowledge or activity characterised by a set of concepts and terminology understood by practitioners in that area.

element

An atomic constituent of a model.

enumeration

A list of named values used as the range of a particular attribute type. For example, RGBColor = {red, green, blue}. Boolean is a predefined enumeration with the values {false, true}.

event

The specification of a significant occurrence that has a location in time and space. In the context of state diagrams, an event is an occurrence that can trigger a state transition.

expression

A string that evaluates to a value of a particular type. For example, the expression "(7 + 5 * 3)" evaluates to a value of type number.

extends

A Use Case relationship where one Use Case 'extends' another Use Case. The 'extends' relationship allows the extending Use Case to be generally consistent with the extended Use Case, while allowing it to be specialised. Thus it does similar things, differently.

feature

A property, like an operation or attribute, which is encapsulated within another entity, such as an interface, a class or a datatype.

fire

To execute a transition on a state diagram.

focus of control

A symbol on a sequence diagram that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

generalization

A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed. See: *inheritance*.

guard condition

A condition that resolves to true or false. In a sequence diagram, a guard condition shows a message that is sent because the guard condition is true. In a statechart diagram, a guard condition shows a transition that can only fire if the guard condition is true.

implementation

A definition of how something is constructed or computed. For example, a method is an implementation of an operation.

implementation inheritance

The inheritance of the implementation of a more-specific element. Includes inheritance of the interface. Contrast: *interface inheritance*.

inheritance

The mechanism by which more-specific elements incorporate structure and behaviour of more-general elements related by behaviour. See *generalization*.

instance

An object of a specified class.

interaction

A specification of how messages are sent between instances to perform a specific task. The interaction may be shown in a sequence or collaboration diagram.

interaction diagram

A generic term that applies to several types of diagrams that emphasise object interactions. These include collaboration diagrams, sequence diagrams and activity diagrams

interface

A declaration of a collection of operations. Used by a class that implements the interface.

interface inheritance

The inheritance of an interface (a declaration of a collection of operations). Does not include inheritance of the implementation. Contrast: *implementation inheritance*.

layer

A specific way of grouping packages in a model at the same level of abstraction.

link

A connection between a pair of (or more) objects. An instance of an association. See: *association*.

message

A specification of a communication between instances that conveys information with the expectation that activity will ensue. The receipt of a message is normally considered an event.

metaclass

A class whose instances are classes. Metaclasses are typically used to construct metamodels.

metamodel

A model that defines the language for expressing a model.

method

1. The implementation of an operation. It specifies the algorithm or procedure that affects the results of an operation. 2. A software-development process, such as UML.

model

A definition of a system.

model element

An element that is an abstraction drawn from the system being modelled.

multiple inheritance

A variation of generalization in which a class may have more than one superclass.

multiplicity

A specification of the number of links that an object may have for a given association.

n-ary association

An association among three or more classes. Each instance of the association is an n-tuple of values from the respective classes. Contrast: *binary association*.

name

A string used to identify a model element.

namespace

A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning. See: *name*.

node

A node is a run-time physical object that represents a computational resource, generally having at least a memory and often processing capability as well. Run-time objects and components may reside on nodes.

object

An entity with a well-defined boundary and identity that encapsulates state and behaviour. State is represented by attributes and relationships. Behaviour is represented by operations, methods and state machines. An object is an instance of a class. See: *class*, *instance*.

object diagram

A diagram that encompasses objects and their links and values at a point in time.

object lifeline

A line in a sequence diagram that represents the existence of an object over a period of time. See: *sequence diagram*.

operation

A definition of a piece of behaviour on a class. The operation may be called on an object of the class which will execute the relevant method.

organisation

A stereotype of actor. Used for interaction by any of an organised collection of people.

package

A general-purpose mechanism for organising elements into groups. Packages may be nested within other packages. Typically, packages are used to organise Use Cases, and separately, classes. A system may be thought of as a single high-level package, with everything else in the system contained in it.

parameter

The specification of a variable that can be changed, passed or returned. A parameter may include a name, type and direction. Parameters are used for operations, messages and events.

parameterised element

An element of a class which serves as a place-holder until the parameterised class is instantiated, when it is replaced by the actual value required.

persistent object

An object that exists after the process that created it has terminated.

process

An executing program.

property

A named value that denotes a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others may be user defined. See: *tagged value*.

pseudo-state

A vertex in a state machine that has the form of a state, but doesn't behave as a state. Pseudo-states include initial, final and history vertices.

qualifier

An association attribute used to index an object across an association.

reference

An attribute of a class that facilitates navigation to associated objects.

relationship

A semantic connection among model elements. Examples of relationships include associations and generalizations.

repository

The place where all the information regarding the system is kept. Includes all information from the models. Essentially, each diagram is a view on selected portions of the repository.

requirement

A desired feature, property or behaviour of a system.

role

A stereotype of actor. Used for human interaction. Many people may participate in the same role.

scenario

A specific instance of a Use Case. It gives a real example of an actor using the system, with real values and a specific system state.

sequence diagram

A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in the interaction, and the sequence of messages exchanged. Unlike a collaboration diagram, a sequence diagram highlights time sequences. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways. See: *collaboration diagram*.

signature

The name, parameters, parameter types, parameter defaults and return type of an operation.

specification

A declarative description of what something is or does. Contrast: *implementation*.

state

A period of time in which an object has a particular set of attribute and link values, or is performing some action, or is waiting for an event.

statechart diagram

A diagram for a class that shows how objects of that class behave at run time. It shows which events the object responds to and highlights the different behaviour of an object in response to an event, depending on the state of the object.

stereotype

A type of modelling element that extends the semantics of the metamodel. Stereotypes must extend classes in the metamodel. Stereotypes may extend the semantics. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of three extensibility mechanisms in UML. See: *constraint*, *tagged value*.

subclass

In a generalization relationship, a class that is a specialisation of another class, the superclass. See: *generalization*. Contrast: *superclass*.

substate

A state that is part of a composite state. See: *concurrent state, disjoint state*.

superclass

In a generalization relationship, a class that is a generalization of another class, the subclass. See: *generalization*. Contrast: *subclass*.

swimlane

A partition on activity diagrams for organising responsibilities for actions. They often correspond to organisational units in a business model.

synchronous action

A request where the sending object pauses to wait for results. Contrast: *asynchronous action*.

system

1. A collection of connected units that are organised to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. 2. A stereotype of actor. Used for systems that are mainly software rather than hardware.

tagged value

The explicit definition of a property as a name-value pair. In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML, others may be user defined. Tagged values are one of three extendibility mechanisms in UML. See: *constraint, stereotype*.

thread

A part of a process that may run concurrently with other threads from the same process.

timing mark

A denotation for the time at which an event or message occurs. Timing marks may be used in constraints or guard conditions.

transient object

An object that does not need to be remembered after its creating program has terminated.

transition

A connection between two states that indicates that if an object in the first state receives a specified event, then the object will transition to the second state. Extra conditions can be added (guard conditions) and actions that will be performed when the transition is made can be specified.

Use Case

The specification of a single piece of functionality of a system. It describes the sequence of actions that a system can perform while interacting with actors of the system.

Use Case diagram

A diagram that shows the relationships among actors and Use Cases within a system.

Use Case model

A model that describes a system's functional requirements in terms of Use Cases.

uses

A Use Case relationship where one Use Case 'uses' another Use Case. The 'uses' relationship causes the used Use Case to be run completely within the 'uses' Use Case.

visibility

Used to describe whether an item can be accessed from other items. A visibility on a class within a package determines whether the class can be used from other packages. A visibility on an attribute or operation within a class determines whether the attribute or operation can be accessed or called from other classes. Visibility may be public, protected, private, implementation or other, depending on language.

Intentionally blank for any new words you might want to add!