Rational Software

Glossary

Version 2003.06.00

Revision History

Date	Version	Description	Author
June, 1999	V4.2	Initial creation	Shawn Siemers
November, 2001	V2002	Final release	Shawn Siemers
December, 2002	V2003.06.00	Final release	Alex Kutsick

Table of Contents

Glossary

1. Introduction

This document contains definitions for the terms used with the Mastering Object-Oriented Analysis and Design with UML course. The majority of the definitions were taken from the Rational Unified Process. Some were take directly from the course materials, or paraphrased from the UML User's Guide[1], or a combination of both. Graphics were included where they helped explain the definition.

2. Definitions

2.1 Abstract Class

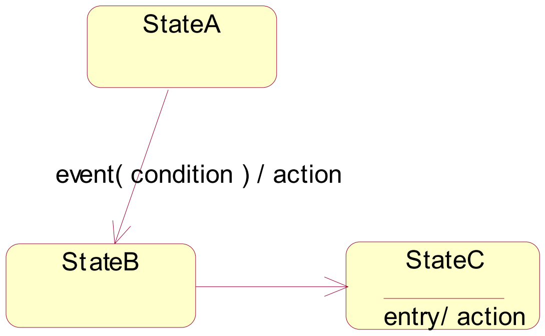
A class that cannot be directly instantiated; the opposite of a concrete class.

2.2 Abstraction

The essential characteristics of an entity that distinguish it from all other kind of entities and thus provide crisply-defined boundaries relative to the perspective of the viewer.

2.3 Action

An action is an operation that is associated with a transition. Actions conceptually take an insignificant amount of time to complete, and are considered non-interruptible. Action names are shown on the transition arrow preceded by a slash.



2.4 Active Class

An active class is a class that “owns” it’s own thread of execution and can initiate control activity, contrasted with passive classes that can only be acted upon. Active classes may execute in parallel (that is, concurrently) with other active classes.

2.5 Activity

A unit of work a worker may be asked to perform.

2.6 Activity Diagram

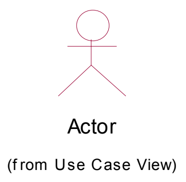
A diagram that models the flow from activity to activity. It is used to model the dynamic view of a system.

2.7 Activity State

The performance of an activity or step within the workflow.

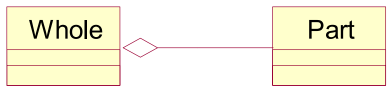
2.8 Actor

Someone or something outside the system or business that interacts with the system or business.



2.9 Aggregation

An association that models a whole-part relationship between an aggregate (the whole) and its parts.



2.10 Analysis

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

2.11 Analysis Class

Analysis classes handle primarily functional requirements, and model objects from the "problem" domain.

2.12 Analysis Mechanism

An architectural mechanism used early in the design process, during the period of discovery when key classes and subsystems are being identified. Typically analysis mechanisms capture the key aspects of a solution in a way that is implementation independent. Analysis mechanisms are usually unrelated to the problem domain, but instead are "computer science" concepts. They provide specific behaviors to a domain-related class or component, or correspond to the implementation of cooperation between classes and/or components. They may be implemented as a framework. Examples include mechanisms to handle persistence, inter-process communication, error or fault handling, notification, and messaging, to name a few.

2.13 Architectural Mechanism

An architectural mechanism represents a common solution to a frequently encountered problem. They may be patterns of structure, patterns of behavior, or both.

2.14 Architecture

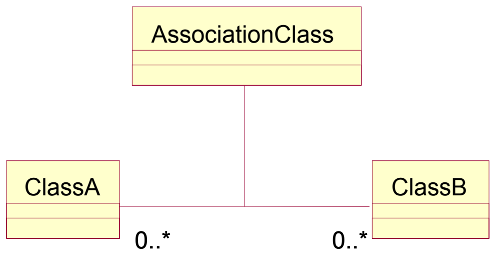
The highest level concept of a system in its environment [IEEE]. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces.

2.15 Association

A relationship that models a bi-directional semantic connection among instances.

2.16 Association Class

An association class is a class that is connected to an association. It is a full-fledged class and can contain attributes, operations and other associations. Association classes allow you to store information about the relationship itself. Such information is not appropriate, or does not belong, within the classes at either end of the relationship.



2.17 Attribute

An attribute defined by a class represents a named property of the class or its objects. An attribute has a type that defines the type of its instances.

2.18 Behavior

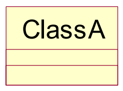
The observable effects of an event, including its results.

2.19 Boundary Class

A class used to model communication between the system's environments and its inner workings.

2.20 Class

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



2.21 Class Diagram

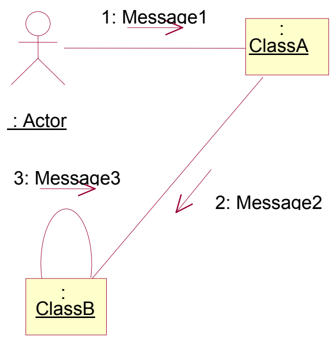
A diagram that shows a set of classes, interfaces, and collaborations and their relationships; class diagrams address the static design view of a system; a diagram that shows a collection of declarative (static) elements.

2.22 Collaboration

A society of roles and other elements that work together to provide some cooperative behavior that’s bigger than the sum of all its parts; the specification of how an element, such as a use case or an operation, is realized by a set of classifiers and associations playing specific roles and used in a specific way.

2.23 Collaboration Diagram

A collaboration diagram describes a pattern of interaction among objects; it shows the objects participating in the interaction by their links to each other and the messages they send to each other.

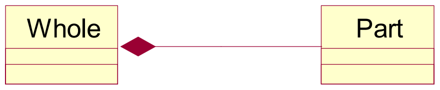


2.24 Component

A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces.

2.25 Composition

A form of aggregation with strong ownership and coincident lifetime of the parts by the whole.



2.26 Connections

Communication mechanisms that can be described by physical mediums (Ethernet, fiber optic cable) or software protocol (TCP/IP).

2.27 Control Class

A class used to model behavior specific to one, or a several use cases.

2.28 Dependency

A semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (dependent thing).

2.29 Deployment Diagram

A diagram that shows the configuration of run time processing nodes and the components that live on them. A deployment diagram addresses the static view of the system.

Work station
Server

2.30 Deployment View

An architectural view that describes one or several system configurations; the mapping of software components (tasks, modules) to the computing nodes in these configurations.

2.31 Derived Attribute

An attribute whose value may be calculated based on the value of other attribute(s).

2.32 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.

2.33 Design Model

An object model describing the realization of use cases; serves as an abstraction of the implementation model and its source code.

2.34 Design Mechanism

An architectural mechanism used during the design process, during the period in which the details of the design are being worked-out. They are related to associated analysis mechanisms, of which they are additional refinements. A design mechanism assumes some details of the implementation environment, but it is not tied to a specific implementation (as is an implementation mechanism). For example, the analysis mechanism for inter-process communication may be refined by several design mechanisms for interprocess communication (IPC): shared memory, function-call-like IPC, semaphore-based IPC, and so on. Each design mechanism has certain strengths and weaknesses; the choice of a particular design mechanism is determined by the characteristics of the objects using the mechanism.

2.35 Encapsulation

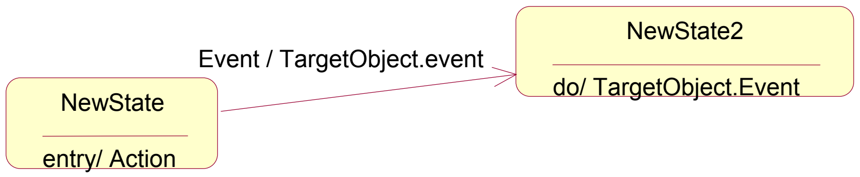
The physical localization of features (for example, properties, behaviors) into a single black box abstraction that hides their implementation (and associated design decisions) behind a public interface. Encapsulation is also referred to as information hiding.

2.36 Entity Class

A class used to model information that has been stored by the system, and the associated behavior. A generic class reused in many use cases, often with persistent characteristics. An entity class defines a set of entity objects, which participate in several use cases and typically survive those use cases.

2.37 Event

An event is an occurrence that happens at some point in time. In a statechart, an event is an occurrence of a stimulus that can trigger a state transition.



2.38 Façade Class

A class that acts as the entry point to a subsystem.

2.39 Focus of Control

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

2.40 Forward Engineering

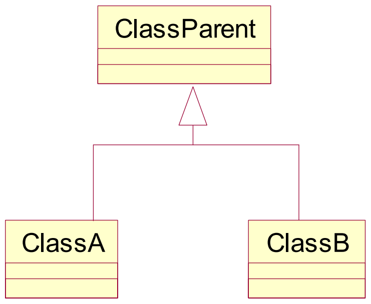
The process of transforming a model into code through a mapping to a specific implementation language.

2.41 Framework

A micro-architecture that provides an incomplete template for applications within a specific domain

2.42 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 can be used where the more general element is allowed.



2.43 Hierarchy

Any ranking or ordering of abstractions into a tree-like structure. Kinds: aggregation hierarchy, class hierarchy, containment hierarchy, inheritance hierarchy, partition hierarchy, specialization hierarchy, type hierarchy. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

2.44 Implementation Mechanism

An architectural mechanism used during the implementation process. They are refinements of design mechanisms, and specify the exact implementation of the mechanism. For example, one particular implementation of the inter-process communication analysis mechanism is a shared memory design mechanism utilizing a particular operating system's shared memory function calls. Concurrency conflicts (inappropriate simultaneous access to shared memory) may be prevented using semaphores, or using a latching mechanism, which in turn rest upon other implementation mechanisms.

2.45 Implementation View

An architectural view that describes the organization of the static software elements (code, data, and other accompanying artifacts) on the development environment, in terms of both packaging, layering, and configuration management (ownership, release strategy, and so on). In the Unified Process it is a view on the implementation model.

2.46 Inheritance

The mechanism that makes generalization possible; a mechanism for creating full class descriptions out of individual class segments.

2.47 Instance

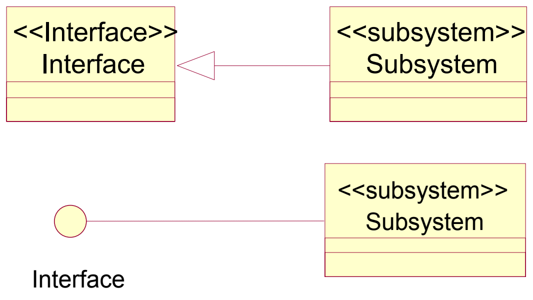
A concrete manifestation of an abstraction; an entity to which a set of operations can be applied and that has a state that stores the effects of the operations.

2.48 Interaction diagram

A diagram that shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them; interaction diagrams address the dynamic view of a system; a generic term that applies to several types of diagrams that emphasize object interactions, including collaboration diagrams, sequence diagrams, and activity diagrams.

2.49 Interface

A collection of operations that are used to specify a service of a class or a component.



2.50 Iteration

A distinct set of activities with a baseline plan and evaluation criteria that results in a release, either internal or external.

2.51 Lifeline

The lifeline represents the existence of the object at a particular time. You can use a lifeline to model both class and object behavior. Usually, a lifeline represents all objects of a certain class.

2.52 Link

A semantic connection among objects; an instance of an association.

2.53 Logical View

An architectural view that describes the main classes in the design of the system: major business-related classes, and the classes that define key behavioral and structural mechanisms (persistency, communications, fault-tolerance, user-interface). In the Unified Process, the logical view is a view of the design model.

2.54 Message

A specification of a communication between objects that conveys information with the expectation that activity will ensue.

2.55 Method

(1) A regular and systematic way of accomplishing something; the detailed, logically ordered plans or procedures followed to accomplish a task or attain a goal. (2) UML 1.1: The implementation of an operation, the algorithm, or the procedure that effects the results of an operation.

2.56 Modularity

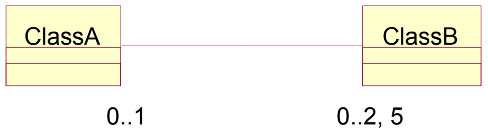
The logical and physical decomposition of things (for example, responsibilities and software) into small, simple groupings (for example, requirements and classes, respectively), which increase the achievements of software-engineering goals.

2.57 Multiple Inheritance

A semantic variation of generalization in which an object may belong directly to more than one class.

2.58 Multiplicity

A specification of the range of allowable cardinalities that a set may assume.

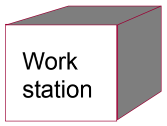


2.59 Navigability

The navigability property on a role indicates that it is possible to navigate from a associating class to the target class using the association.

2.60 Node

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.



2.61 Object

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

2.62 Object-orientation (OO)

The Rational Unified Process supports object-oriented techniques. Each model is object-oriented. Rational Unified Process models are based on the concepts of objects and classes and the relationships among them, as they use the UML as its common notation.

2.63 Object Technology

A set of principles (abstraction, encapsulation, polymorphism) guiding software construction, together with languages, databases, and other tools that support those principles. (*Object Technology - A Manager's Guide*, Taylor, 1997.)

2.64 Operation

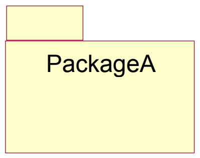
A service that can be requested from an object to effect behavior.

2.65 Operation signature

The name and parameters of an operation.

2.66 Package

A mechanism for organizing elements into groups.



2.67 Pattern

A scheme for describing design fragments or collections of class templates so that they can be configured and reused.

2.68 Polymorphism

Polymorphism is the ability to define a single interface with multiple implementations.

2.69 Process

(1) Any thread of control that can logically execute concurrently with other processes. (2) A set of partially ordered steps intended to reach a goal; in software engineering the goal is to build a software product or to enhance an existing one; in process engineering, the goal is to develop or enhance a process model; corresponds to a business use case in business engineering.

2.70 Process View

An architectural view that describes the concurrent aspect of the system: tasks (processes) and their interactions.

2.71 Property

A named value denoting a characteristic of an element.

2.72 Realization

A semantic relationship between classifiers, in which one classifier specifies a contract that another classifier guarantees to carry out.

2.73 Responsibility

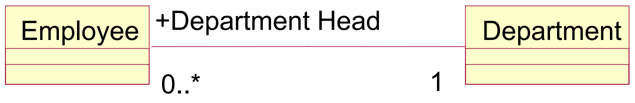
A contract or obligation of a type or class.

2.74 Reverse Engineering

The process of transforming code into a model through a mapping from a specific implementation language.

2.75 Role

The behavior of an entity participating in a particular context.

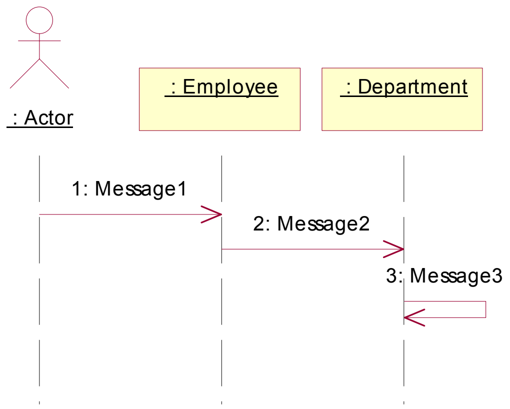


2.76 Scenario

A described use-case instance, a subset of a use case.

2.77 Sequence Diagram

A diagram that describes a pattern of interaction among objects, arranged in a chronological order; it shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other.



2.78 Single Inheritance

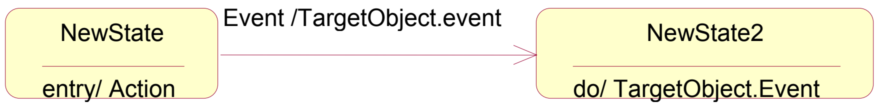
A semantic variation of generalization in which a child may have only one parent.

2.79 State

A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

2.80 Statechart Diagram

A statechart diagram shows a state machine, that is, a behavior that specifies the sequences of states that an object goes through during its life in response to events, together with its responses and actions.



2.81 Stereotype

A meta-classification of an element. Stereotypes have semantic implications which can be specified for every specific stereotype value.

2.82 Subsystem

A model element which has the semantics of a package, such that it can contain other model elements, and a class, such that it has behavior. (The behavior of the subsystem is provided by classes or other subsystems it contains). A subsystem realizes one or more interfaces, which define the behavior it can perform.

2.83 Swimlane

A partition on an interaction diagram for organizing responsibilities for actions.

2.84 Thread

An independent computation executing within an the execution environment and address space defined by an enclosing operating system process.

2.85 Transition

A transition is a change from an originating state to a successor state as a result of some stimulus.

2.86 Unified Modeling Language (UML)

A language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.

2.87 Use Case

A use case defines a set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor. A use-case class contains all main, alternate flows of events related to producing the 'observable result of value'. Technically, a use-case is a class whose instances are scenarios.

2.88 Use Case Model

A model of what the system is supposed to do and the system environment.

2.89 Use Case Realization

A use-case realization describes how a particular use case is realized within the design model, in terms of collaborating objects.

2.90 Use Case View

An architectural view that describes how critical use cases are performed in the system, focusing mostly on architecturally significant components (objects, tasks, nodes). In the Unified Process, it is a view of the use-case model.

2.91 Utility Class

A class that contains a collection of free subprograms.

2.92 Visibility

How a name can be seen and used by others.

2.93 Visual Modeling

A way of thinking about problems using models organized around real-world ideas.