IBM / Rational software

Glossary

Version 2004.06.00

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
June, 2004	V2004.06.00	Final release	Alex Kutsick

Table of Contents

Glossary

1. Introduction

This document contains definitions for terms used in the Essentials of Visual Modeling and Mastering Object-Oriented Analysis and Design with UML courses. Many of the definitions are from the Rational Unified Process and some are from the Unified Modeling Language Reference Manual, 2nd edition, by James Rumbaugh, Ivar Jacobson, Grady Booch, Addison-Wesley, Boston, 2005. Other information was taken directly from the course materials or paraphrased from the UML User's Guide. Graphics were included where they helped to explain the definition.

2. Definitions

2.1 Abstract Class

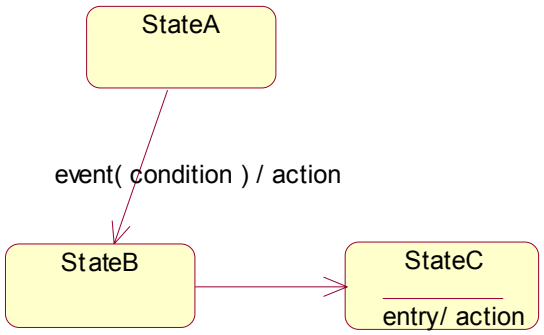
An abstract class is a class that cannot be instantiated—that is, it may not have direct instances. It is the opposite of a concrete class.

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

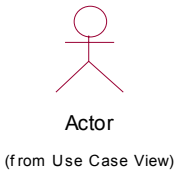
An activity diagram shows the decomposition of an activity into its constituents.

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. It is shown by a hollow diamond at the end of the path attached to the aggregate class.

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. 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 Assembly Connector

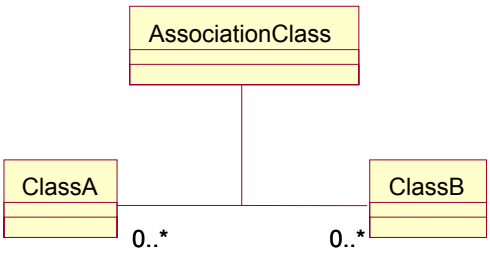
A connector between two elements (parts or ports) in the internal implementation specification of a structured classifier or component.

2.16 Association

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

2.17 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.



2.18 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.

2.19 Ball

A provided interface relationship shown by a small circle, or a ball, attached to a classifier by a line.

2.20 Behavior

The observable effects of an event that includes results.

2.21 Boundary Class

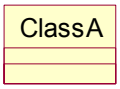
A class used to model communication between the system's environments and its inner workings.

2.22 Choice

A node in a state machine at which dynamic evaluations of subsequent guard conditions is made.

2.23 Class

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



2.24 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.

2.25 Classifier

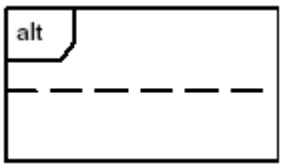
A model element that describes behavioral and structural features. Kinds of classifiers include actor, association, class, collaboration, component, data type, interface, node, signal, subsystem, and use case.

2.26 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.

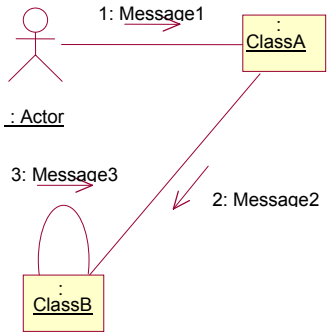
2.27 Combined Fragment

A construct within an interaction that comprises an operator keyword and one or more interaction operands, each of which is a fragment of an interaction. It is shown as a nested region within a sequence diagram. If the fragment has more than one subfragment, horizontal dashed lines separate them.



2.28 Communication Diagram

A communication 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.



2.29 Component

A modular part of a system that hides its implementation behind a set of external interfaces. Within a system, components satisfying the same interfaces may be substituted freely.

2.30 Component Diagram

A diagram that shows the definition, internal structure, and dependencies of component types. There is no sharp line between component diagrams and general class diagrams.

2.31 Composite Structure Diagram

A composite structure diagram shows the internal structure (including parts and connectors) of a structured classifier or collaboration. It defines the parts of a system and the communication relationships between them.

2.32 Composition

A composition is a stronger form of association in which the composite has sole responsibility for managing its parts— such as their allocation and deallocation. A filled diamond on the composite end shows it. An object at most may belong to one composition.

2.33 Concrete Class

A generalizable element (such as a class) that can be directly instantiated. Of necessity, its implementation must be fully specified. For a class, all its operations must be implemented (by the class or an ancestor).

2.34 Concurrency

Concurrency is the tendency for things to happen at the same time in a system.

2.35 Connector

The connection of two structured parts within a structured classifier or a collaboration; a specification of an contextual association that applies only in a certain context, such as the objects within a classifier or objects satisfying a collaboration.

2.36 Control Class

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

2.37 Delegation Connector

A connector between an external port of a structured class or component and an internal part. Connections to the external port are treated as going to the element at the other end of the delegation connector.

2.38 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).

2.39 Deployment

The assignment of software artifacts to physical nodes during execution.

2.40 Deployment Diagram

A diagram that shows the configuration of run-time processing nodes and the artifacts that live on them. A deployment diagram may be at the class level or the instance level.

2.41 Deployment Specification

A detailed specification of the parameters of the deployment of an artifact to a node. A deployment specification is shown as a rectangle symbol with the keyword «deploymentSpec».

2.42 Deployment View

A view that shows the nodes in a distributed system, the artifacts that are stored on each node, and the components and other elements that the artifacts manifest.

2.43 Derived Attribute

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

2.44 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.

2.45 Design Model

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

2.46 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.

2.47 Device

A physical computational resource with processing capability upon which artifacts may be deployed for execution. A node annotated with the stereotype <<device>> notates a device.

2.48 Encapsulation

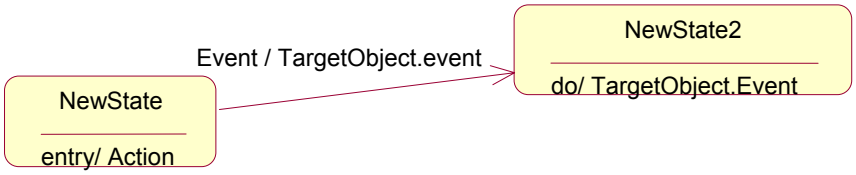
The physical localization of features (for example, properties, behaviors) into a single black box abstraction that hides their implementation (and associated design decisions) behind a public interface. Encapsulation is also referred to as information hiding.

2.49 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.

2.50 Event

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



2.51 Event Occurrence

The occurrence of an event during the execution of a system, e.g., call event, signal event, time event, change event. An event occurrence is not explicitly shown as a separate concept. It is usually shown by the intersection of a message arrow and a lifeline.

2.52 Execution Environment

A kind of deployment node that represents a particular kind of execution platform, such as an operating system, a workstation engine, a database management system, and so on. Also (and more commonly) used informally to describe the context within which execution of a model occurs. A node annotated with the stereotype <<ExecutionEnvironment>> notates an execution environment.

2.53 Execution Occurrence

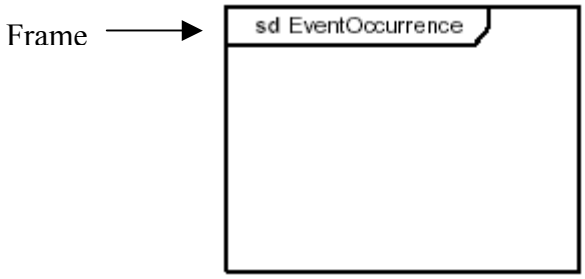
The execution of an activity, operation, or other behavior unit within an interaction. An execution represents the period during which an object performs a behavior either directly or through a subordinate behavior.

2.54 Forward Engineering

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

2.55 Frame

A diagram is presented as a frame containing graphical contents. The frame names the diagram and establishes its extent. It is drawn as a rectangle with a small pentagon (called the name tag) in the upper left corner.



2.56 Framework

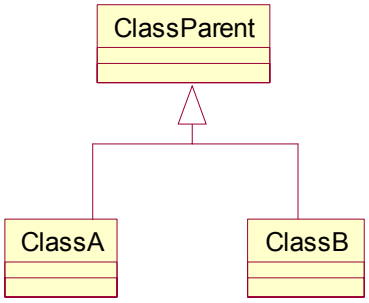
A micro-architecture that provides an incomplete template for applications within a specific domain

2.57 Gate

A connection point in an interaction or interaction fragment for a message that comes from or goes to outside the interaction or fragment.

2.58 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.



2.59 General Ordering

A constraint in an interaction where the time of one event occurrence precedes the time of another event occurrence.

2.60 Guard Condition

The guard is expressed as a Boolean constraint on values available to test at the time of messaging, i.e., the guard determines whether a transition may fire.

2.61 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.)

2.62 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.

2.63 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.

2.64 Inheritance

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

2.65 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.

2.66 Interaction

A specification of how messages are exchanged between objects or other instances over time to perform a task. An interaction is defined in a context, which may be a classifier, a collaboration, or some other grouping of connected parts.

2.67 Interaction Diagram

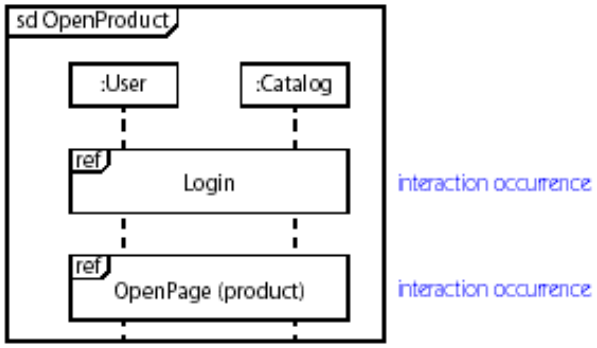
A diagram that shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them; interaction diagrams address the dynamic view of a system; a generic term that applies to several types of diagrams that emphasize object interactions, including communication diagrams, sequence diagrams, timing diagrams and the interaction overview diagrams. The Interaction Diagram is a generic term for focusing on messaging [interaction] between objects. As such, there is no one graphic for an Interaction Diagram.

2.68 Interaction Fragment

A structural piece of an interaction.

2.69 Interaction Occurrence

A reference to an interaction within the definition of another interaction.



2.70 Interaction Operand

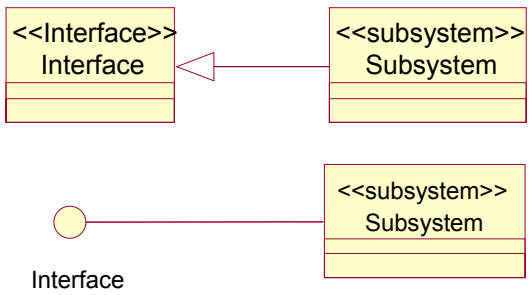
A structural piece of a combined fragment; a subfragment.

2.71 Interaction Overview Diagram

A diagram that depicts interactions through a variant of activity diagrams in such a way to promote an overview of the control flow. It focuses on the overview of the flow of control where each node can be an interaction diagram.

2.72 Interface

A declaration of a coherent set of public features and obligations; a contract between providers and consumers of services.



2.73 Iteration

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

2.74 Iteration Expression

A specification of the range of number of iterations of a loop.

2.75 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.

2.76 Link

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

2.77 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, communication, fault-tolerance, user-interface). In the Unified Process, the logical view is a view of the design model.

2.78 Manifestation

The physical implementation of a model element as an artifact. A manifestation is shown as a dependency arrow from an artifact to a model element. The keyword «manifest» is placed on the arrow.

2.79 Message

The conveyance of information from one object (or other instance) to another as part of an interaction within a context. A message may be a signal or the call of an operation. The sending and the receipt of a message are event occurrences.

2.80 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.

2.81 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.

2.82 Multiple Inheritance

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

2.83 Multiplicity

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

2.84 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.

2.85 Node

A run-time physical object that represents a computational resource, generally having at least memory and often processing capability. Run-time artifacts may be deployed on nodes.

2.86 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.

2.87 Object Diagram

A diagram that encompasses objects and their relationships at a given point in time. An object diagram may be considered a special case of a class diagram or a communication diagram.

2.88 Object Lifeline

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

2.89 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.

2.90 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.)

2.91 Operation

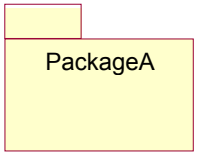
A service that can be requested from an object to effect behavior.

2.92 Operation Signature

The name and parameters of an operation.

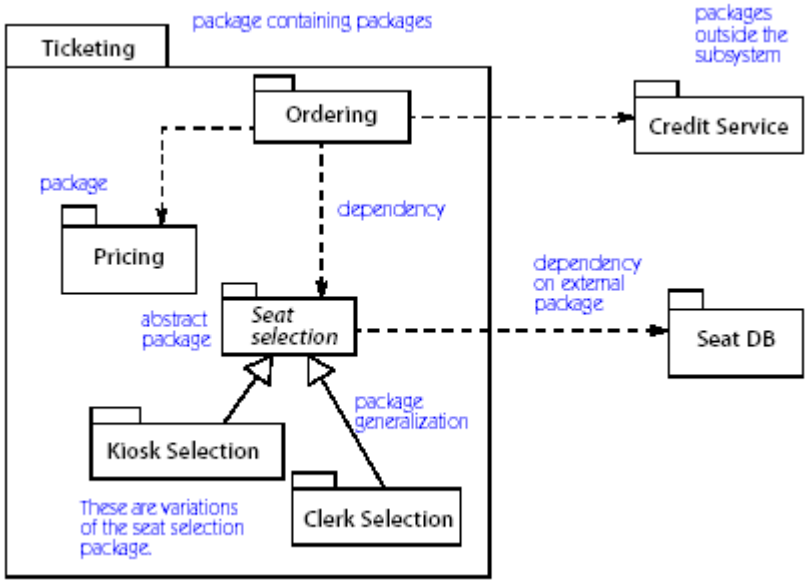
2.93 Package

A general-purpose mechanism for organizing elements into groups, establishing ownership of elements, and providing unique names for referencing elements.



2.94 Package Diagram

A diagram that depicts how model elements are organized into packages and the dependencies among them, including package imports and package extensions.



2.95 Package Import

A directed relationship that adds the names of elements to a namespace.

2.96 Partitions

The organization of activities into distinct regions. Organize activities in a model according to their responsibility—for example, group all the activities handled by one business organization. Partitions are separated by lines in the diagram.

2.97 Pattern

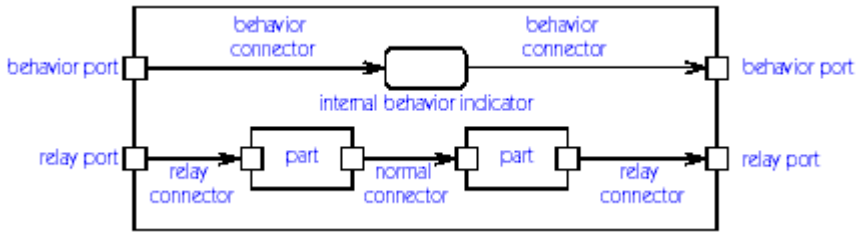
A scheme for describing design fragments or collections of class templates so that they can be configured and reused.

2.98 Polymorphism

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

2.99 Port

A structural feature of a classifier that encapsulates interaction between the contents of the classifier and its environment. A port is shown as a small square straddling the boundary of a classifier rectangle. The name of the port is placed near the square.



2.100 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.

2.101 Process View

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

2.102 Property

A named value denoting a characteristic of an element.

2.103 Provided Interface

An interface that declares the services that a classifier offers to provide to anonymous requestors. A provided interface relationship is shown by a small circle, or a ball, attached to a classifier by a line. Alternately, a provided interface can be shown using realization notation.

2.104 Realization

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

2.105 Relationship

An abstract concept that specifies some kind of connection between elements. Examples of relationships include associations and generalizations.

2.106 Required Interface

A required interface is the complementary relationship of a provided interface where a classifier requires the services described in the interface. A required interface relationship is shown by a small half circle, or a socket, attached to a classifier by a line. Alternately, a required interface can be shown using dependency notation.

2.107 Responsibility

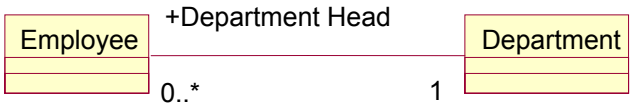
A contract or obligation of a type or class.

2.108 Reverse Engineering

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

2.109 Role

The behavior of an entity participating in a particular context. Role names are not underlined when only a role name is needed and no instance reference is implied.

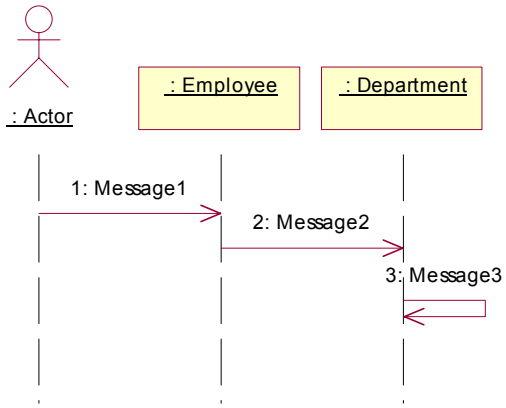


2.110 Scenario

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

2.111 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.



2.112 Single Inheritance

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

2.113 Socket

A required interface relationship is shown by a small half circle, or a socket, attached to a classifier by a line.

2.114 State

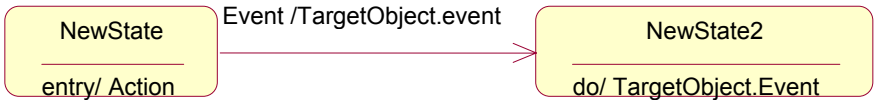
A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

2.115 State Machine

A specification of the sequences of states that an object or an interaction goes through in response to events during its life, together with its responsive effects (action and activity). A state machine is attached to a source class, collaboration, or method and specifies the behavior of the instances of the source element.

2.116 State Machine Diagram

A state machine 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.



2.117 Stereotype

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

2.118 Stored Procedures

A stored procedure is executable code that runs under the RDBMS. Stored procedures provide the ability to perform database-related actions on the server without having to transfer data across a network.

2.119 Structured Class

A class containing parts or roles that form its structure and realize its behavior.

2.120 Structure Diagram

A form of diagram that depicts the elements in a specification that is irrespective of time. Class diagrams and component diagrams are examples of structure diagrams.

2.121 Structured Part

Within a structured classifier, an element that represents an object or set of objects within a contextual relationship.

2.122 Subsystem

A large unit of decomposition for a system. It is modeled as a stereotype of component with the keyword <<subsystem>>.

2.123 Thread

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

2.124 Time Constraint

Expressed as a time interval, it can refer to a single event occurrence or to the time interval between two occurrences.

2.125 Timing Diagram

An interaction diagram that shows the change in state or condition of a lifeline over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli. It is an optional diagram designed to specify the time constraints on messages sent and received in the course of an interaction.

2.126 Transaction

Transactions define a set of operation invocations that are atomic: either all or none of them are performed.

2.127 Transition

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

2.128 Unified Modeling Language (UML)

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

2.129 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.

2.130 Use-Case Diagram

A diagram that shows the relationships among actors and use cases within a system.

2.131 Use-Case Model

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

2.132 Use-Case Realization

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

2.133 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.

2.134 Utility Class

A class that contains a collection of free subprograms.

2.135 Visibility

How a name can be seen and used by others.

2.136 Visual Modeling

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