## Glossary

This glossary defines many of the terms used in Enterprise Architect.

[Glossary 1](#_Toc525769116)

[~A~ 2](#_Toc525769117)

[~B~ 5](#_Toc525769118)

[~C~ 7](#_Toc525769119)

[~D~ 11](#_Toc525769120)

[~E~ 14](#_Toc525769121)

[~F~ 16](#_Toc525769122)

[~G~ 18](#_Toc525769123)

[~H~ 19](#_Toc525769124)

[~I~ 20](#_Toc525769125)

[~J~ 22](#_Toc525769126)

[~L~ 23](#_Toc525769127)

[~M~ 24](#_Toc525769128)

[~N~ 27](#_Toc525769129)

[~O~ 28](#_Toc525769130)

[~P~ 30](#_Toc525769131)

[~Q~ 34](#_Toc525769132)

[~R~ 35](#_Toc525769133)

[~S~ 38](#_Toc525769134)

[~T~ 43](#_Toc525769135)

[~U~ 45](#_Toc525769136)

[~V~ 47](#_Toc525769137)

## ~A~

Terms starting with A

|  |  |
| --- | --- |
| Name | Detail |
| Abstract Class | A Class that cannot be directly instantiated. |
| Abstraction | The essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer. |
| Action | The specification of an executable statement that forms an abstraction of a computation procedure. An action typically results in a change in the state of the system, and can be realized by sending a message to an object or modifying a connector or a value of an attribute. |
| Action Sequence | An expression that resolves to a sequence of actions. |
| Action State | A State that represents the execution of an atomic Action, typically the invocation of an operation. |
| Activation | The execution of an Action. |
| Active Class | A Class whose instances are Active Objects. When instantiated, an Active Class controls its execution. Rather than being invoked or activated by other objects, it can operate standalone, and define its own thread of behavior. |
| Active Object | An object that owns a thread and can initiate control activity. An instance of active Class. |
| Activity | Defines the bounds for the structural organization that contains a set of basic or fundamental behaviors. It can used to model procedural type application development for system design through to modeling business processes in organizational structures and workflow. |
| Activity Diagram | A diagram used to model procedural type application development for system design through to modeling business processes in organizational structures and workflow. |
| Activity Graph | A special case of a StateMachine that is used to model processes involving one or more classifiers. |
| Actor (Class) | A coherent set of roles that users of Use Cases play when interacting with these Use Cases. An Actor has one role for each Use Case with which it communicates. |
| Actual Parameter | A binding for a parameter that resolves to a run-time instance. |
| Aggregate (Class) | 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. |
| 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. |
| Analysis Diagram | A diagram used to capture high level business processes and early models of system behavior and elements. It is less formal than some other diagrams, but provides a good means of capturing the essential business characteristics and requirements. |
| Analysis Time | Refers to something that occurs during an analysis phase of the software development process. |
| Architecture | The organizational structure and associated behavior of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include Classes, Components and subsystems. |
| Argument | A binding for a parameter that resolves to a run-time instance. |
| Artifact | A physical piece of information that is used or produced by a business or development process. Examples of Artifacts include models, source files, scripts, and binary executable files. An Artifact can constitute the implementation of a deployable component. |
| Assembly | A connector that bridges the required interface of a component with the provided interface of a second component. |
| Association | The semantic relationship between two or more classifiers that specifies connections among their instances. |
| Association Class | A model element that has both Association and Class properties. An Association Class can be seen as an Association that also has Class properties, or as a Class that also has Association properties. |
| Association End | The endpoint of an Association, which connects the Association to a classifier. |
| Attribute | A feature within a classifier that describes a range of values that instances of the classifier can hold. |
| Auxiliary Class | A stereotyped Class that supports another more central or fundamental Class, typically by implementing secondary logic or control flow. Auxiliary Classes are typically used together with focus Classes, and are particularly useful for specifying the secondary business logic or control flow of components during design. |

## ~B~

Terms starting with B

|  |  |
| --- | --- |
| Name | Detail |
| Behavior | The observable effects of an operation or event, including its results. |
| Behavioral Diagram | A diagram that depicts the behavioral features of a system or business process.  Behavioral diagrams include Activity diagrams, StateMachine diagrams, Communication diagrams, Interaction Overview diagrams, Sequence diagrams, Timing diagrams and Use Case diagrams. |
| Behavioral Feature | A dynamic feature of a model element, such as an operation or method. |
| Behavioral Model aspect | A model aspect that emphasizes the behavior of the instances in a system, including their methods, collaborations, and state histories. |
| Binary Association | An Association between two Classes; a special case of an N-ary Association. |
| Binding | The creation of a model element from a template by supplying arguments for the parameters of the template. |
| Bookmark | A marker in a Rich Text Format document that enables you to link inner sections of a document into a master document (using the Word 'Insert File' function). |
| Boolean | An enumeration whose values are true and false. |
| Boolean Expression | An expression that evaluates to a boolean value. |
| Boundary | 1. A stereotyped Object that models some system boundary – typically a user interface screen; it is:  - Used in the conceptual phase to capture user interaction with the system at a screen level (or some  other boundary interface type)  - Often used in Sequence and Robustness (Analysis) diagrams  - The View in the Model-View-Controller Pattern 2. A System Boundary element used to delineate a particular part of the system. |

## ~C~

Terms starting with C

|  |  |
| --- | --- |
| Name | Detail |
| C++ | An object-oriented programming language based on the earlier 'C' language. |
| Call | An Action state that invokes an operation on a classifier. |
| Cardinality | The number of elements in a set. |
| CASE | Computer Aided Software Engineering. A tool designed for the purpose of modeling and building software systems. |
| Child | In a Generalization relationship, the specialization of another element, the parent. |
| Choice | A pseudostate used to compose complex transitional paths, where the outgoing transition path is decided by dynamic, run-time conditions determined by the actions performed by the StateMachine on the path leading to the choice. |
| Class | A description of a set of objects that share the same attributes, operations, methods, relationships and semantics. A Class can use a set of interfaces to specify collections of operations it provides to its environment. |
| Class Diagram | A diagram that shows a collection of declarative (static) model elements, such as Classes, types, and their contents and relationships. |
| Classification | The assignment of an object to a classifier. |
| Classifier | A mechanism that describes behavioral and structural features. Classifiers include Interfaces, Classes, datatypes, and components. |
| Client | A classifier that requests a service from another classifier. |
| Collaboration | The specification of how an operation or classifier, such as a Use Case, is realized by a set of classifiers and Associations playing specific roles used in a specific way. The Collaboration defines an interaction. |
| Collaboration Diagram | Used pre-UML 2.0. Now called a Communication diagram. |
| Collaboration Use | Uses an occurrence to apply a Pattern defined by a Collaboration to a specific situation. |
| Combined Fragment | A combined fragment reflects a piece or pieces of interaction (called interaction operands) controlled by an interaction operator, whose corresponding boolean conditions are known as interaction constraints. It appears graphically as a transparent window, divided by horizontal dashed lines for each operand. |
| Comment | An annotation attached to an element or a collection of elements. A comment, or note, has no semantics. |
| Communication Diagram | A diagram that shows the interactions between elements at run-time in much the same manner as a Sequence diagram. However, Communication diagrams are used to visualize inter-object relationships, while Sequence diagrams are more effective at visualizing processing over time. |
| Compile Time | Refers to something that occurs during the compilation of a software module. |
| Component | A modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. A Component is typically specified by one or more classifiers (such as implementation Classes) that reside on it, and can be implemented by one or more artifacts (such as binary, executable or script files). |
| Component Diagram | A diagram that shows the organizations and dependencies among Components. |
| Composite (Class) | A Class that is related to one or more Classes by a Composition relationship. |
| Composite State | A State that consists of either concurrent (orthogonal) substates or sequential (disjoint) substates. |
| Composite Structure Diagram | A diagram that reflects the internal collaboration of Classes, Interfaces, or Components to describe a functionality. **Composite Structure** diagrams are similar to Class diagrams, except that they model a specific usage of the structure. |
| Composition | A form of Aggregation that requires that a part instance be included in at most one Composite at a time, and that the Composite object is responsible for the creation and destruction of the parts. Composition can be recursive. |
| Concrete Class | A Class that can be directly instantiated. |
| 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. |
| Concurrent Substate | A substate that can be held simultaneously with other substates contained in the same composite State. |
| Connector | A logical link between model elements. Can be structural, dynamic or possessive. |
| Constraint | 1. A semantic condition or restriction. Certain constraints are predefined in the UML, others can be user defined. Constraints are one of three extensibility mechanisms in UML. 2. A rule or condition that applies to some element. It is often modeled as a pre- or post- condition. |
| Container | 1. An instance that exists to contain other instances, and that provides operations to access or iterate over its contents (for example, arrays, lists, sets). 2. A component that exists to contain other components. |
| Containment Hierarchy | A namespace hierarchy consisting of model elements, and the containment relationships that exist between them. A containment hierarchy forms a graph. |
| Context | A view of a set of related modeling elements for a particular purpose, such as specifying an operation. |
| Continuation | A Continuation is used in seq and alt combined fragments, to indicate the branches of continuation an operand follows. |
| Control | A stereotyped Object that represents a controlling entity or manager. A Control organizes and schedules other activities and elements. It is the controller of the Model-View-Controller Pattern. |
| Control Flow | A connector linking two nodes in an activity diagram. Control Flow connectors start a node's activity when the preceding node's action is finished. |

## ~D~

Terms starting with D

|  |  |
| --- | --- |
| Name | Detail |
| Database Schema | The description of a database structure. It defines tables and fields and the relationship between them. |
| Datastore | An element used to define permanently stored data. A token of data that is stored in the Datastore is stored permanently. A token of data that comes out of the Datastore is a copy of the original data. The tokens imported are kept for the life of the Activity in which it exists. |
| Datatype | A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations. |
| Decision | An element of an Activity diagram that indicates a point of conditional progression: if a condition is true, then processing continues one way, if not, then another. |
| Defining Model (MOF) | The model on which a repository is based. Any number of repositories can have the same defining model. |
| Delegate | A connector that defines the internal assembly of a component's external ports and interfaces. Using a Delegate connector wires the internal workings of the system to the outside world, by a delegation of the external interfaces' connections. |
| Delegation | The ability of an object to issue a message to another object in response to a message. Delegation can be used as an alternative to inheritance.  Contrast: inheritance |
| Dependency | A relationship between two modeling elements, in which a change to one modeling element (the independent element) affects the other modeling element (the dependent element). |
| Deployment | A type of Dependency relationship that indicates the deployment of an artifact onto a node or executable target. |
| 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. |
| Deployment Specification | Specifies parameters guiding deployment of an artifact, as is common with most hardware and software technologies. |
| 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 is to be implemented. During design, strategic and tactical decisions are made to meet the required functional and quality requirements of a system. |
| Design Time | Refers to something that occurs during a design phase of the software development process. |
| 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 14 diagram types, and Enterprise Architect extends these with seven more. **Add-Ins**, technologies and profiles can provide further diagram types. |
| Diagram Gate | A simple graphical way to indicate the point at which messages can be transmitted into and out of Interaction Fragments. |
| Diagram View | The Enterprise Architect workspace area where the UML diagrams are displayed. |
| Disjoint Substate | A substate that cannot be held simultaneously with other substates contained in the same composite State. |
| Distribution Unit | A set of objects or components that are allocated to a process or a processor as a group. A distribution unit can be represented by a run-time composite or an Aggregate. |
| Domain | An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area. |
| Dynamic Classification | A semantic variation of Generalization in which an object can change its classifier. |

## ~E~

Terms starting with E

|  |  |
| --- | --- |
| Name | Detail |
| Element | 1. An atomic constituent of a model. 2. A model object of any type, such as Class, Component, Node or Object. |
| Endpoint | Used in Interaction diagrams to reflect a lost message in sequence. |
| Entity | A store or persistence mechanism that captures the information or knowledge in a system. It is the Model in the Model-View-Controller Pattern. |
| Entry Action | An action executed upon entering a State in a StateMachine regardless of the transition taken to reach that State. |
| Entry Point | Used to define where external states can enter a Sub Machine. |
| 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 values from the set {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 transition. |
| Exception Handler | An element that defines the group of operations to carry out when an exception occurs. |
| Exit Action | An action executed upon exiting a State in a StateMachine regardless of the transition taken to exit that State. |
| Exit Point | Used in SubMachine states and StateMachines to denote the point where the machine is exited and the transition sourcing this exit point, for SubMachines, is triggered. Exit points are a type of pseudostate used in the **StateMachine diagram**. |
| Export | In the context of Packages, to make an element visible outside its enclosing namespace. |
| Expose Interface | A toolbox icon that is a graphical way to depict the required and supplied interfaces of a Component, Class or Part. |
| 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.  A relationship from an extension Use Case to a base Use Case, specifying how the behavior defined for the extension Use Case augments (subject to conditions specified in the extension) the behavior defined for the base Use Case. The behavior is inserted at the location defined by the extension point in the base Use Case. The base Use Case does not depend on performing the behavior of the extension Use Case. |
| Extend | A connector used to indicate that an element extends the behavior of another. Extensions are used in Use Case models to indicate one Use Case (optionally) extends the behavior of another. |

## ~F~

Terms starting with F

|  |  |
| --- | --- |
| Name | Detail |
| Facade | A stereotyped Package containing only references to model elements owned by another Package; it is used to provide a 'public view' of some of the contents of a Package. |
| Feature | A property, like an operation or attribute, that is encapsulated within a classifier such as an Interface, Class, or Datatype. |
| Final | A pseudo-state that indicates an end. |
| Final State | A special kind of State signifying that the enclosing composite State or the entire StateMachine is completed. |
| Fire | To execute a State transition. |
| Flow Final | An element that depicts an exit from the system, as opposed to the Activity Final, which represents the completion of the activity. |
| Focus Class | A stereotyped Class that defines the core logic or control flow for one or more auxiliary Classes that support it.  Focus Classes are typically used together with one or more auxiliary Classes, and are particularly useful for specifying the core business logic or control flow of components during design. |
| 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. |
| Forward Engineering | The process of generating source code from the UML model. |
| Fork | Used in StateMachine diagrams as pseudostates.  With respect to StateMachine diagrams, a Fork pseudostate signifies that its incoming transition comes from a single State, and it has multiple outgoing transitions. |
| Framework | A stereotyped Package containing model elements that specify a reusable architecture for all or part of a system.  Frameworks typically include Classes, Patterns or templates.  When frameworks are specialized for an application domain, they are sometimes referred to as Application frameworks. |

## ~G~

Terms starting with G

|  |  |
| --- | --- |
| Name | Detail |
| Generalizable Element | A model element that can participate in a Generalization relationship. |
| 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. |
| Guard Condition | A condition that must be satisfied in order to enable an associated transition to fire. |

## ~H~

Terms starting with H

|  |  |
| --- | --- |
| Name | Detail |
| History State | There are two types of History pseudo-states defined in UML: shallow History and deep History.  A shallow History sub-state is used to represent the most recently active sub-state of a composite State.  A deep History sub-state, in contrast, reflects the most recent active configuration of the composite State. |

## ~I~

Terms starting with I

|  |  |
| --- | --- |
| Name | Detail |
| Implementation | A definition of how something is constructed or computed. For example, a Class is an implementation of a type, a method is an implementation of an operation. |
| Implementation Class | A stereotyped Class that specifies the implementation of a Class in some programming language (for example, C++, Smalltalk, Java) in which an instance can not have more than one Class. An Implementation Class is said to realize a type if it provides all of the operations defined for the type with the same behavior as specified for the type's operations. |
| Implementation Inheritance | The inheritance of the implementation of a more general element. Includes inheritance of the interface. |
| Import | In the context of Packages, a dependency that shows the Packages whose Classes can be referenced within a given Package (including Packages recursively embedded within it). |
| Include | A relationship from a base Use Case to an inclusion Use Case, specifying how the behavior for the base Use Case contains the behavior of the inclusion Use Case. The behavior is included at the location that is defined in the base Use Case. The base Use Case depends on performing the behavior of the inclusion Use Case, but not on its structure (that is, attributes or operations). |
| Inheritance | The mechanism by which more specific elements incorporate the structure and behavior of more general elements related by behavior. |
| Initial State | A pseudo-state used to denote the default state of a composite State; there can be one initial vertex in each region of the composite State. |
| Instance | An entity that has a unique identity, a set of operations that can be applied to it, and a state that stores the effects of the operations. |
| Interaction | A specification of how stimuli are sent between instances to perform a specific task. The interaction is defined in the context of a collaboration. |
| Interaction Diagram | A generic term that applies to several types of diagrams that emphasize object interactions. These include Timing diagrams, Sequence diagrams, Interaction Overview diagrams and Communication diagrams. |
| Interaction Occurrence | A reference to an existing interaction element. Interaction occurrences are visually represented by a frame, with ref in the frame's title space. The diagram name is indicated in the frame contents. |
| Interaction Overview Diagram | A diagram that visualizes the cooperation between other Interaction diagrams to illustrate a control flow serving an encompassing purpose. As Interaction Overview diagrams are a variant of Activity diagrams, most of the diagram notation is similar, as is the process in constructing the diagram. |
| Interface | A named set of operations that characterize the behavior of an element. |
| Interface Inheritance | The inheritance of the interface of a more general element. Does not include inheritance of the implementation. |
| Internal Transition | A transition signifying a response to an event without changing the state of an object. |
| Interrupt Flow | An Enterprise Architect-defined toolbox icon used to define the exception handler and interruptible activity region concepts. |

## ~J~

Terms starting with J

|  |  |
| --- | --- |
| Name | Detail |
| Java | A fully object-oriented, cross platform language based on elements from Smalltalk, C++ and other OO languages. |
| Join | Used in StateMachine diagrams and in Activity diagrams to synchronize multiple flows. |
| Junction | Junction pseudo-states are used to design complex transitional paths. A Junction can be used to combine, or merge, multiple paths into a shared transition path or to split an incoming path into multiple paths. |

## ~L~

Terms starting with L

|  |  |
| --- | --- |
| Name | Detail |
| Layer | The organization of classifiers or Packages at the same level of abstraction. A layer represents a horizontal slice through an architecture, whereas a partition represents a vertical slice. |
| Lifeline | An individual participant in an interaction (that is, Lifelines cannot have multiplicity). A Lifeline represents a distinct connectable element. |
| Link | A semantic connector among a tuple of objects.  An instance of an Association. |
| Link End | An instance of an Association end. |
| Local Path | A relative path on a local machine, enabling developers to store shared source code in machine specific directories, but still generate and synchronize code. |

## ~M~

Terms starting with M

|  |  |
| --- | --- |
| Name | Detail |
| Maintenance | The support of a software system after it is deployed. |
| Manifest | A relationship that indicates that the artifact source embodies the target model element. Stereotypes can be added to Enterprise Architect to classify the type of manifestation of the model element. |
| Message | Messages indicate a flow of information, or transition of control, between elements. Messages are used by Communication diagrams, Sequence diagrams, Interaction Overview diagrams and Timing diagrams. |
| Message Endpoint | An element that defines an endpoint of a Lifeline, such as a State or Value Lifeline in a Timing diagram. |
| Message Label | Used for messages sent between Lifelines to make the diagram appear less cluttered. Labels with the same name indicate that a message can be interrupted. |
| Metaclass | A Class whose instances are Classes. Metaclasses are typically used to construct metamodels. |
| Metafile | A vector-based image format native to Windows. Supports high detail and excellent scaling. Typically used for saving diagram images for placement in documents. Comes in Placeable (an older format) and Enhanced (current standard format). |
| Meta-Metamodel | A model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model. |
| Metamodel | A model that defines the language for expressing a model. |
| Meta-Object | A generic term for all meta-entities in a meta-modeling language. For example, meta-types, meta-classes, meta-attributes, and meta-associations. |
| Meta-Object Facility (MOF) | An Object Management Group (OMG) standard. MOF originated in the UML, when the OMG required a Meta-Modeling architecture to define the UML. MOF is designed as a four-layered architecture. |
| Method | The implementation of an operation. It specifies the algorithm or procedure associated with an operation. |
| Model (MOF) | An abstraction of a physical system with a certain purpose.Usage note: In the context of the MOF specification, which describes a meta-metamodel, the meta-metamodel is for brevity frequently referred to simply as the model. |
| Model Aspect | A dimension of modeling that emphasizes particular qualities of the metamodel. For example, the structural model aspect emphasizes the structural qualities of the metamodel. |
| Model Elaboration | The process of generating a repository type from a published model. Includes the generation of interfaces and implementations which enables repositories to be instantiated and populated based on, and in compliance with, the model elaborated. |
| Model Element (MOF) | An element that is an abstraction drawn from the system being modeled. In the MOF specification model elements are considered to be meta-objects. |
| Model Library | A stereotyped Package containing model elements that are intended to be reused by other Packages. A model library differs from a profile in that a model library does not extend the metamodel using stereotypes and tagged definitions. A model library is analogous to a Class library in some programming languages. |
| Modeling Time | Refers to something that occurs during the modeling phase of the software development process. It includes analysis time and design time.Usage note: When discussing object systems, it is often important to distinguish between modeling-time and run-time concerns. |
| Module | A software unit of storage and manipulation. Modules include source code modules, binary code modules and executable code modules. |
| MOF | Meta-Object Facility, an Object Management Group (OMG) standard. MOF originated in the UML, when the OMG required a Meta-Modeling architecture to define the UML. MOF is designed as a four-layered architecture. |
| Multiple Classification | A semantic variation of Generalization in which an object can belong directly to more than one classifier. |
| Multiple Inheritance | A semantic variation of Generalization in which a type can have more than one supertype. |
| Multiplicity | A specification of the range of enableable cardinalities that a set can assume. Multiplicity specifications can be given for roles within Associations, Parts within Composites, repetitions and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers. |
| Multi-Valued (MOF) | A model element with multiplicity defined whose Multiplicity Type:: upper attribute is set to a number greater than one. The term multi-valued does not pertain to the number of values held by, for example, an attribute or parameter, at any point in time. |

## ~N~

Terms starting with N

|  |  |
| --- | --- |
| Name | Detail |
| Name | A string used to identify a model element. |
| Namespace | A part of the model in which the names can be defined and used. Within a namespace, each name has a unique meaning. |
| 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. |
| Nesting | A connector used as an alternative membership notation to indicate nested members within an element; for example, a Package that has nested members. The nested members of a Package could also be shown inside the Package rather than linked by the Nesting connector. |
| Node | A classifier that represents a run-time computation resource, which generally has at least a memory and often processing capability. Run-time objects and components can reside on nodes. |

## ~O~

Terms starting with O

|  |  |
| --- | --- |
| Name | Detail |
| 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, methods and StateMachines. An Object is an instance of a Class. |
| Object Diagram | A diagram that encompasses objects and their relationships at a point in time. An Object diagram can be considered as a special case of a Class diagram or Communication diagram. |
| Object Flow | A sub type of the State flow or transition. It implies the passing of an object instance between elements at run-time. |
| Object Flow State | A state in an Activity graph that represents the passing of an object from the output of actions in one State to the input of actions in another State. |
| Object Lifeline | A line in a Sequence diagram that represents the existence of an object over a period of time. |
| Object Management Group (OMG) | The standards body responsible for the UML specification and management. |
| Occurrence | A relationship that indicates that a Collaboration represents a classifier. An Occurrence connector is drawn from the collaboration to the classifier. |
| Operation | A service that can be requested from an object to effect behavior. An operation has a signature, which could restrict the actual parameters that are possible. |

## ~P~

Terms starting with P

|  |  |
| --- | --- |
| Name | Detail |
| Package | 1. A namespace, as well as an element that can be contained in other Packages' namespaces. Packages can own or merge with other Packages, and their elements can be imported into a Package's namespace. 2. A logical container of model elements. It groups elements and can also contain other Packages.   The OMG UML specification (UML Superstructure Specification, v2.1.1, p.109) states:  A Package is used to group elements, and provides a namespace for the grouped elements.  A package is a namespace for its members, and can contain other packages. Only packageable elements can be owned members of a package. By virtue of being a namespace, a package can import either individual members of other packages, or all the members of other packages.  In addition a package can be merged with other packages.  Note that Packages own model elements and are the basis for configuration control, storage and access control. Each element can be directly owned by a single Package, so the Package hierarchy is a strict tree. However, Packages can reference other Packages, modeled by using one of the stereotypes «import» and «access» of Permission dependency, so the usage network is a graph. Other kinds of dependencies between Packages usually imply that one or more dependencies among the elements exist.  A Package is represented by the common folder icon - a large rectangle with a small rectangle (a 'tab') attached to the left side on top. |
| Package Diagram | Used to reflect the organization of Packages and their elements, and provide a visualization of their corresponding namespaces. |
| Package Import | A Package import relationship is drawn from a source Package to a Package whose contents are imported. Private members of a target Package cannot be imported. |
| Package Merge | Indicates a relationship between two Packages whereby the contents of the target Package are merged with those of the source Package. Private contents of a target Package are not merged. |
| Parameter | The specification of a variable that can be changed, passed, or returned. A parameter can include a name, type, and direction. Parameters are used for operations, messages and events. |
| Parameterized element | The descriptor for a Class with one or more unbound parameters. |
| Parent | In a generalization relationship, the generalization of another element, the child. |
| Part | A run-time instance of a Class or Interface. |
| Participate | The connection of a model element to a relationship or to a reified relationship. For example, a Class participates in an Association, an Actor participates in a Use Case. |
| Partition | 1. activity graphs: A portion of an activity graph that organizes the responsibilities for actions. 2. architecture: A set of related classifiers or Packages at the same level of abstraction or across layers in a layered architecture. A partition represents a vertical slice through an architecture, whereas a layer represents a horizontal slice. |
| Pattern | A template collaboration. |
| Persistent Object | An object that exists after the process or thread that created it has ceased to exist. |
| Physical System | 1. The subject of a model. 2. A collection of connected physical units, which can include software, hardware and people, that are organized to accomplish a specific purpose. A physical system can be described by one or more models, possibly from different viewpoints. |
| Port | Defines the interaction between a classifier and its environment. Interfaces controlling this interaction can be depicted using the 'Expose Interface' toolbox icon. |
| Postcondition | A constraint that must be true at the completion of an operation. |
| Precondition | A constraint that must be true when an operation is invoked. |
| Primitive Type | A pre-defined basic datatype without any substructure, such as an integer or a string. |
| Process | 1. A heavyweight unit of concurrency and execution in an operating system. Thread, which includes heavyweight and lightweight processes. If necessary, an implementation distinction can be made using stereotypes. 2. A software development process - the steps and guidelines by which to develop a system. 3. To execute an algorithm or otherwise handle something dynamically. |
| Product | A physical piece of information that is produced by a business or development process. Examples of products include models, source files, scripts, and binary executable files. An product can constitute the implementation of a deployabComponentle component. |
| Profile | A stereotyped Package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. A profile can also specify model libraries on which it depends and the metamodel subset that it extends. |
| Project Browser | The workspace window where the model contents are displayed in 'tree' format. Displays structures such as Packages, diagrams and model elements. |
| Projection | A mapping from a set to its subset. |
| Property | A named value denoting a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others can be user defined. |
| Pseudostate | A vertex in a StateMachine that has the form of a State, but doesn't behave as a State. Pseudostates include Initial and History vertices. |
| Published Model (MOF) | A model that has been frozen, and that becomes available for instantiating repositories and for support in defining other models. A frozen model's model elements cannot be changed. |

## ~Q~

Terms starting with Q

|  |  |
| --- | --- |
| Name | Detail |
| Qualifier | An Association attribute or tuple of attributes whose values partition the set of objects related to an object across an Association. |

## ~R~

Terms starting with R

|  |  |
| --- | --- |
| Name | Detail |
| Realize | A source object realizes the destination object.  Realize is used to express traceability and completeness in the model – a business process or requirement is realized by one or more Use Cases which are in turn realized by some Classes which in turn are realized by a Component, and so on. |
| Receive (A message) | The handling of a stimulus passed from a sender instance. |
| Receive | An element used to define the acceptance or receipt of a request; movement on to the next action occurs until it has received what is defined. |
| Receiver (Object) | The object handling a stimulus passed from a sender object. |
| Reception | A declaration that a classifier is prepared to react to the receipt of a signal. |
| Recursion | A type of message used in Sequence diagrams to indicate a recursive function. |
| Reference | 1. A denotation of a model element. 2. A named slot within a classifier that facilitates navigation to other classifiers. |
| Region | UML 2.x supports both Expansion Regions and Interruptible Activity Regions.   * An Expansion Region defines the bounds of a region consisting of one or more sets of input collections, where an input collection is a set of elements of the same type * An Interruptible Activity Region contains Activity nodes - when a token leaves an interruptible region, this terminates all of the region's tokens and behaviors |
| Refinement | A relationship that represents a fuller specification of something that has already been specified at a certain level of detail; for example, a design Class is a refinement of an analysis Class. |
| Relationship | A semantic connection among model elements; examples of relationships include Associations and Generalizations. |
| Repository | A facility for storing object models, interfaces and implementations. |
| Represents | A connector that indicates a Collaboration Use is used in a classifier; the connector is drawn from the Collaboration Use to its owning classifier. |
| Requirement | A required feature, property or behavior of a system (external requirement). |
| Responsibility | A contract or obligation of a classifier (internal requirement). |
| Reuse | The use of a pre-existing artifact. |
| Reverse Engineering | The process of importing source code into the model as standard UML model objects (such as Classes, attributes and operations). |
| Rich Text Format | A standard mark-up language for creating word processor documents, frequently associated with Microsoft Word. |
| Robustness Diagram | Enterprise Architect supports business process modeling extensions from the UML business process model profile; Robustness diagrams are used in ICONIX. |
| Role | 1. The named detail and rules associated with one end of an association; a role can indicate name, constraints, multiplicity and collection details. 2. The named specific behavior of an entity participating in a particular context; a role can be static (such as an Association end) or dynamic (such as a Collaboration role). |
| Role Binding | The mapping between a Collaboration Use's internal roles and the respective parts required to implement a specific situation; the associated parts can have properties defined to enable the binding to occur, and the collaboration to take place. |
| Run Time | The period of time during which a computer program executes. |

## ~S~

Terms starting with S

|  |  |
| --- | --- |
| Name | Detail |
| Scenario | 1. A specific sequence of actions that illustrates behaviors. A scenario can be used to illustrate an interaction or the execution of a Use Case instance. 2. A sequence of operations carried out in some order to produce a known result. Can apply to Use Cases where it is the equivalent of a Sequence diagram, or to other objects to describe how they are used at run-time. |
| Schema (MOF) | In the context of the MOF, analogous to a Package that is a container of model elements. Schema corresponds to a MOF Package. |
| Self-Message | Reflects a new process or method invoked within the calling Lifeline's operation. It is a specification of a message. |
| Semantic Variation Point | A point of variation in the semantics of a metamodel. It provides an intentional degree of freedom for the interpretation of the metamodel semantics. |
| Send (A message) | The passing of a stimulus from a sender instance to a receiver instance. |
| Sender (Object) | The object passing a stimulus to a receiver object. |
| 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 Communication (Collaboration) diagram, a Sequence diagram includes time sequences but does not include object relationships. A Sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario).  Sequence diagrams and Communication diagrams express similar information, but show it in different ways. |
| Signal | The specification of an asynchronous stimulus communicated between instances. Signals can have parameters. |
| Signature | The name and parameters of a behavioral feature. A signature can include an optional returned parameter. |
| Single Inheritance | A semantic variation of Generalization in which a type can have only one supertype. |
| Single Valued (MOF) | A model element with multiplicity defined is single valued when its Multiplicity Type: upper attribute is set to 1.  The term single-valued does not pertain to the number of values held by, for example, an attribute or parameter at any point in time, since a single-valued attribute (for instance, with a multiplicity lower bound of zero) could have no value. |
| Specification | A declarative description of what something is or does. |
| 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. |
| State Invariant | A condition applied to a Lifeline that must be fulfilled for the Lifeline to exist. |
| StateMachine | A behavior that specifies the sequences of States that an object or an interaction goes through during its life in response to events, together with its responses and actions. |
| StateMachine Diagram | A diagram that illustrates how an element, often a Class, can move between States classifying its behavior, according to transition triggers, constraining guards and other aspects of StateMachine diagrams that depict and explain movement and behavior. |
| State Chart | A diagram that shows a StateMachine. |
| State Continuation | A symbol that serves two different purposes for Interaction diagrams - as State Invariants and as Continuations.  A State Invariant is a condition applied to a Lifeline that must be fulfilled for the Lifeline to exist.  A Continuation is used in seq and alt combined fragments to indicate the branches of continuation that an operand follows. |
| State Lifeline | A State Lifeline follows discrete transitions between States, which are defined along the y-axis of the timeline. Any transition has optional attributes of timing constraints, **duration** constraints and observations. |
| Static Classification | A semantic variation of Generalization in which an object can not change classifier. |
| Stereotype | A new type of modeling element that extends the semantics of the metamodel.  Stereotypes must be based on certain existing types or Classes in the metamodel. Tagged value  Stereotypes can extend the semantics, but not the structure of pre-existing types and Classes. Certain stereotypes are predefined in the UML, others can be user defined. Stereotypes are one of three extensibility mechanisms in UML. |
| Stimulus | The passing of information from one instance to another, such as raising a signal or invoking an operation. The receipt of a signal is normally considered an event. |
| String | A sequence of text characters. The details of string representation depend on implementation, and can include character sets that support international characters and graphics. |
| Structural Diagram | A diagram that depicts the structural elements composing a system or function.  These diagrams can reflect the static relationships of a structure, as do Class or Package diagrams, or run-time architectures, such as Object or **Composite Structure** diagrams.  Structural diagrams include Class diagrams, Composite Structure diagrams, Component diagrams, Deployment diagrams, Object diagrams and Package diagrams. |
| Structural Feature | A static feature of a model element, such as an attribute. |
| Structural Model Aspect | A model aspect that emphasizes the structure of the objects in a system, including their types, Classes, relationships, attributes and operations. |
| Subactivity State | A State in an activity graph that represents the execution of a non-atomic sequence of steps that has some **duration**. |
| Subclass | In a Generalization relationship, the specialization of another Class; the superclass. |
| Submachine State | A State in a StateMachine that is equivalent to a composite State but its contents are described by another StateMachine. |
| Sub-Package | A Package that is contained in another Package. |
| Substate | A State that is part of a composite State. |
| Subsystem | A grouping of model elements that represents a behavioral unit in a physical system.  A subsystem offers interfaces and has operations. In addition, the model elements of a subsystem can be partitioned into specification and realization elements. |
| Subtype | In a Generalization relationship, the specialization of another type; the supertype. |
| Superclass | In a Generalization relationship, the generalization of another Class; the subclass. |
| Supertype | In a Generalization relationship, the generalization of another type; the subtype. |
| Supplier | A classifier that provides services that can be invoked by others. |
| Swimlane | A partition on an Activity diagram for organizing the responsibilities for Actions. Swimlanes typically correspond to organizational units in a business model. |
| Synch | A State used for indicating that concurrent paths of a StateMachine are synchronized. After bringing the paths to a synch state, the emerging transition indicates unison. |
| Synchronize Code | The process of importing and exporting code changes to ensure the model and source code match. |
| System | A top-level subsystem in a model. |
| System Boundary | An element used to delineate a particular part of the system. |

## ~T~

Terms starting with T

|  |  |
| --- | --- |
| Name | Detail |
| Table | A relational table (composed of columns). |
| Tagged Value | The explicit definition of a property as a name-value pair.  In a **Tagged Value**, the name is referred to as the tag. Certain tags are predefined in the UML; others can be user defined.  **Tagged Values** are one of three extensibility mechanisms in UML. |
| Template | The descriptor for a Class with one or more unbound parameters. |
| Terminate | A pseudostate indicating that upon entry of its pseudostate, the StateMachine's execution ends. |
| Thread (of Control) | A single path of execution through a program, a dynamic model, or some other representation of control flow. Also, a stereotype for the implementation of an active object as a lightweight process. |
| Time Event | An event that denotes the time elapsed since the current state was entered. |
| Time Expression | An expression that resolves to an absolute or relative value of time. |
| Timing Diagram | A diagram that defines the behavior of different objects within a time-scale, with visual depictions of those objects changing state and interacting over time. |
| Toolbox | The main toolbar running down the center of Enterprise Architect, from which you can select model elements to insert into diagrams. This is also known as the **Diagram Toolbox** and the Object toolbar. |
| Top Level | A stereotype of Package denoting the top-most Package in a containment hierarchy.  The topLevel stereotype defines the outer limit for looking up names, as namespaces 'see' outwards. For example, opTopLevelsubsystem represents the top of the subsystem containment hierarchy. |
| Trace | A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other. |
| Transient Object | An object that exists only during the execution of the process or thread that created it. |
| Transition | A relationship between two States indicating that an object in the first State performs certain specified actions and enters the second State when a specified event occurs and specified conditions are satisfied. On such a change of State, the transition is said to fire. |
| Type | A stereotyped Class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects.  A type can not contain any methods, maintain its own thread of control, or be nested. However, it can have attributes and associations.  Although an object can have at most one implementation Class, it can conform to multiple different types. |
| Type Expression | An expression that evaluates to a reference to one or more types. |

## ~U~

Terms starting with U

|  |  |
| --- | --- |
| Name | Detail |
| UML | The Unified Modeling Language, a notation and specification for modeling software systems in an Object-Oriented manner.  You can read more about UML on the OMG home page or in our UML Tutorial. |
| UML Diagrams | Diagrams used to model different aspects of the system under development. They include various elements and connectors, all of which have their own meanings and purposes.  UML 2.5 includes 14 diagrams: Use Case diagram, Activity diagram, **StateMachine diagram**, Timing diagram, Sequence diagram, Interaction Overview diagram, Communication diagram, Package diagram, Class diagram, Profile diagram, Object diagram, **Composite Structure** diagram, Component diagram and Deployment diagram. |
| UML Toolbox | The main toolbar running down the center of Enterprise Architect from which you can select model elements to insert into diagrams. This is also known as the **Diagram Toolbox** and the Object toolbar. |
| Uninterpreted | A placeholder for a type or types whose implementation is not specified by the UML. Every uninterpreted value has a corresponding string representation. |
| Usage | A dependency in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation. |
| Use | A connector that indicates that one element requires another to perform some interaction. The Usage relationship does not specify how the target supplier is used, other than that the source client uses it in definition or implementation. |
| Use Case | A UML model element that describes how a user of the proposed system interacts with the system to perform a discrete unit of work. It describes and signifies a single interaction over time that has meaning for the end user (person, machine or other system), and is required to leave the system in a complete state: either the interaction completed or was rolled back to the initial state. |
| Use Case Diagram | A diagram that captures Use Cases and Actor interactions. It describes the functional requirements of the system, the manner in which outside things (Actors) interact at the system boundary, and the response of the system. |
| Use Case Estimation | The technique of estimating project size and complexity based on the number of Use Cases and their difficulty. |
| Use Case Instance | The performance of a sequence of actions being specified in a Use Case. An instance of a Use Case. |
| Use Case Model | A model that describes a system's functional requirements in terms of Use Cases. |
| Utility | A stereotype that groups global variables and procedures in the form of a Class declaration. The utility attributes and operations become global variables and global procedures, respectively.  A utility is not a fundamental modeling construct, but a programming convenience. |

## ~V~

Terms starting with V

|  |  |
| --- | --- |
| Name | Detail |
| Value | An element of a type domain. |
| Value Lifeline | A Lifeline that shows the Lifeline's state across the diagram, within parallel lines indicating a steady state.  A cross between the lines indicates a transition or change in state. |
| Vertex | A source or a target for a transition in a StateMachine; a vertex can be either a State or a pseudostate. |
| View | A projection of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective. |
| View Element | An element that is a textual and/or graphical projection of a collection of model elements. |
| View Projection | A projection of model elements onto view elements.  A View Projection provides a location and a style for each view element. |
| Visibility | An enumeration whose value ('public', 'protected', 'package' or' private') denotes how the model element to which it refers can be seen outside its enclosing namespace. |
| Visual Basic | A rapid application development programming language.  Windows' only scripting language based on COM. |