Appendice D

Glossario

Gran parte del glossario è tratta direttamente dalle specifiche ufficiali OMG dello UML e pertanto la definizione dei termini va inquadrata in tale contesto. Le convenzioni sono le seguenti:

Contrario
Indica un insieme di concetti la cui semantica è antitetica o comunque utilizzata in modo significativamente diverso da quello del concetto illustrato.

Consultare
Rimanda il lettore ad altri concetti utili per una maggiore comprensione della parola definita.

Sinonimo
Il concetto è sostanzialmente equivalente a un altro definito in altra parte.

[contesto]
Indica il contesto cui si riferisce definizione riportata.

A

Aggregazione – *Aggregation*
Forma particolare di associazione che specifica una relazione (tutto-parte, *whole-part*) tra una classe aggregata (tutto) a una parte componente (parte).
Consultare: Composizione.

Aggregazione composta – *Composite aggregation*
Sinonimo: Composizione.

Analisi – *Analysis*
Parte del processo di sviluppo del software il cui obiettivo primario è la costruzione di un modello (detto appunto di analisi) del dominio del problema iniziando a incorporare direttive provenienti dallo spazio delle soluzioni.

API Interfaccia Programma Applicativo – *API Application Programming Interface*
Interfaccia software che permette alle applicazioni di comunicare tra loro. Un'API è costituita da un insieme di costrutti o di statement specificati in un linguaggio di programmazione che possono essere codificati in un programma eseguibile al fine di ottenere specifiche funzioni e servizi forniti dal sottostante sistema operativo o programma di servizio.

Architettura – *Architecture*
Struttura organizzativa e comportamento associato di un sistema. Un'architettura può essere ricorsivamente decom-

posta in parti che interagiscono attraverso apposite interfacce in relazioni che connettono le varie parti e in vincoli sull'assemblaggio delle parti. A loro volta, le parti che interagiscono per mezzo di interfacce includono: classi, componenti e sottosistemi.
Consultare: Classe, Componente, Interfaccia, Sottosistema.

Argomento – *Argument*

Istanza di un parametro. Valore a tempo di esecuzione da sostituire a un parametro.
Sinonimo: Parametro attuale.
Contrario: Parametro.

Aspetto comportamentale del modello – *Behavioral model aspect*

Aspetto del modello che enfatizza il comportamento delle istanze in un sistema, compresi metodi, stati storici e collaborazioni.

Aspetto del modello – *Model aspect*

Dimensione della modellazione che mette in evidenza specifiche qualità. Per esempio, l'aspetto strutturale del modello ne enfatizza le qualità strutturali.

Aspetto strutturale del modello – *Structural model aspect*

Aspetto del modello che enfatizza la struttura degli oggetti in un sistema, considerando tipi, classi, relazioni, attributi e operazioni.

Associazione – *Association*

Relazione semantica tra due o più classificatori che specifica connessioni delle relative istanze.

Associazione binaria – *Binary association*

Associazione tra due classi. Caso particolare dell'associazione *n*-aria.
Consultare: Associazione *n*-aria.

Associazione *n*-aria – n-*ary association*

Particolare versione di associazione che coinvolge tre o più classi. Ciascuna istanza dell'associazione consiste in un *t*-upla di valori delle rispettive classi.
Contrario: Associazione binaria.

Astrazione – *Abstraction*

Caratteristiche essenziali di un'entità che la distinguono da tutti gli altri tipi di entità. Un'astrazione definisce un confine relativo al punto di vista del fruitore.

Attivazione – *Activation*

Esecuzione di un'azione.
Consultare: Azione.

Attività *[processi di sviluppo del software]* – *Activity*

Unità tangibile di lavoro eseguita da un lavoratore in un workflow (flusso di lavoro) che: 1. implica un insieme ben definito di responsabilità per il lavoratore stesso; 2. genera uno o più risultati ben definiti detti manufatti (*artifacts*), a partire da specifici input e 3. rappresenta un'unità di lavoro con confini ben definiti tipicamente riferita in un piano del progetto quando i lavori sono assegnati ai lavoratori.

Attore – *Actor*

Insieme coerente di ruoli che gli utenti dei casi d'uso possono esercitare interagendo con gli stessi. Un attore possiede un ruolo specifico per ogni caso d'uso con il quale interagisce (scambia messaggi).
Consultare: Caso d'uso.

Attributo – *Attribute*

Proprietà interna di un classificatore, identificata da un nome, che descrive un insieme di valori e operazioni da esso eseguibili, che le istanze del classificatore potrebbero mantenere.
Consultare: Classificatore.

Azione – *Action*

Specifica di un comando (*statement*) eseguibile che costituisce un'astrazione di una procedura di calcolo. Il risultato di un'azione, tipicamente, comporta la variazione dello stato del sistema e può essere realizzato inviando un opportuno messaggio a un oggetto, modificando una relazione, aggiornando il valore di un attributo.

Azione d'entrata – *Entry action*

Azione da eseguire all'atto dell'entrata in uno stato di una macchina a stati indipendentemente dalla transazione che ha portato a raggiungere lo stato stesso.

Azione d'uscita – *Exit action*

Azione eseguita all'atto dell'uscita da uno stato di una macchina a stati indipendentemente dalla transazione che ne ha generato il passaggio di stato.
Consultare: Stato.

B

Baseline – *Baseline*
Insieme di manufatti rivisti e approvati che rappresentano una base concordata per evoluzioni future e sviluppo e possono essere modificati esclusivamente attraverso un processo formale come quello di gestione della configurazione e variazioni.

Beta testing – *Beta testing*
Processo di test che precede il rilascio. Tipicamente viene eseguito da un insieme selezionato di utenti finali. Spesso si ottiene dall'utilizzo di una versione pilota installata direttamente presso l'organizzazione degli utenti finali.

Binding – *Binding*
Creazione di un elemento del modello a partire da un template, specificando i parametri richiesti dal template stesso.
Consultare: Template.

Booleano – *Boolean*
Istanza di un tipo enumerato i cui unici valori previsti sono `true` e `false`.
Consultare: Tipo enumerato.

Build – *Build*
Versione eseguibile del sistema tipicamente focalizzata su una sua parte. I processi di sviluppo iterativi e incrementali prevedono la consegna del sistema finale attraverso la realizzazione di una successione di build.

C

Caratteristica – *Feature*
Caratteristica, come un'operazione o un attributo, incapsulata in un classificatore, come un'interfaccia, una classe o un tipo di dato.
Consultare: Classe, Interfaccia, Tipo di dato

Caratteristica strutturale – *Structural feature*
Caratteristica statica di un elemento del modello come un attributo.

Cardinalità – *Cardinality*
Numero di elementi in un insieme.
Contrario: Molteplicità.

Casi d'uso business – *Business use case*
Modello dei casi d'uso focalizzato sull'area business, in cui ognuno dei singoli casi d'uso mostra le azioni eseguite dal "sistema business" per fornire un risultato osservabile al relativo attore business.

Caso d'uso *[class]* – *Use case*
Specifica di una sequenza di azioni, incluse varianti, che un sistema (o un'altra entità) può eseguire interagendo con gli attori del sistema.
Consultare: Istanza di casi d'uso.

Caso di test – *Test case*
Specificazione di uno scenario di test del sistema in cui si stabilisce cosa verificare, quale situazione iniziale predisporre, quali input introdurre e quali output attendersi.

Centrato sull'architettura *[processi di sviluppo del software]* – *Architecture-centric*
Indica una strategia che comporta che l'architettura del sistema sia utilizzata come manufatto primario per concettualizzare, costruire, gestire e far evolvere il sistema in costruzione.

Chiamata – *Call*
Stato di azione che invoca un'operazione di uno specifico classificatore.
Consultare: Stato di azione.

Classe – *Class*
Descrizione di un insieme di oggetti che condividono stessi attributi, operazioni, metodi, relazioni e semantica. Una classe può utilizzare una serie di interfacce al fine di specificare insiemi di operazioni esposte al proprio ambiente.
Consultare: Interfaccia.

Classe aggregata – *Aggregate class*
Classe che rappresenta il tutto in una relazione di aggregazione (tutto-parte, *whole-part*).
Consultare: Aggregazione.

Classe associazione – *Association class*
Elemento del modello che possiede sia le proprietà della associazione, sia quelle del classificatore. Una classe associazione può essere immaginata come un'associazione dotata delle proprietà della classe o viceversa (una classe che ha proprietà dell'associazione).
Consultare: Associazione, Classe.

Classe astratta – *Abstract class*
Classe che non può essere istanziata direttamente. È presente almeno un metodo astratto da definire in opportune classi specializzanti. È utilizzata per definire un

comportamento comune condiviso da un insieme di sottoclassi.
Consultare: Sottoclasse.
Contrario: Classe concreta.

Classe attiva – *Active class*

Classe le cui istanze sono oggetti attivi.
Consultare: Oggetto attivo.

Classe ausiliaria – *Auxiliary class*

Stereotipo di classe utilizzata per indicare una classe che ne supporta un'altra più centrale o importante. Le classi ausiliarie, tipicamente, realizzano un flusso o una logica secondaria e tendono a essere utilizzate in associazione a classi focalizzate (*focus*). Risultano particolarmente utili per specificare la logica business ausiliaria o il controllo ausiliare dei componenti durante il relativo disegno.
Consultare: Classe focalizzata.

Classe composta – *Composite class*

Classe associata a una o più classi attraverso la relazione di composizione.
Consultare: Composizione.

Classe concreta – *Concrete class*

Classe che può essere istanziata direttamente (non possiede alcuna dichiarazione di metodo astratto).
Contrario: Classe astratta.

Classe focalizzata – *Focus class*

Stereotipo di classe utilizzata per indicare una classe che definisce la parte fondamentale della logica o del flusso di esecuzione per una più classi ausiliarie da essa supportate. Le classi focalizzate sono, tipicamente, utilizzate insieme a diverse classi ausiliarie e risultano particolarmente utili per specificare la logica business fondamentale o la parte principale di controllo di componenti durante il relativo disegno.
Consultare: Classi ausiliarie.

Classe implementazione – *Implementation class*

Stereotipo dell'elemento class che specifica l'implementazione di una specifica classe per mezzo di un linguaggio di programmazione (per esempio Java, C++, SmallTalk, …) in cui le cui istanze non possono avere più di una classe. Si dice che una classe implementazione realizza un tipo se fornisce tutte le operazioni previste dal tipo stesso e se realizza il comportamento così come definito per le operazioni del tipo.
Consultare: Tipo.

Classificatore – *Classifier*

Meccanismo che descrive proprietà strutturali e comportamentali. Specializzazioni del classificatore sono classi, interfacce, tipi di dati, componenti, …

Classificazione – *Classification*

Assegnazione di un oggetto al relativo classificatore.
Consultare: Classificazione dinamica, Classificazione statica, Classificazione multipla.

Classificazione dinamica – *Dynamic classification*

Variazione semantica di generalizzazione in cui un oggetto può cambiare il classificatore di appartenenza.
Contrario: Classificazione statica.

Classificazione multipla – *Multiple classification*

Variazione semantica della generalizzazione nella quale un oggetto può appartenere direttamente a più classificatori.
Consultare: Classificazione statica, Classificazione dinamica.

Classificazione statica – *Static classification*

Variazione semantica della generalizzazione nella quale un oggetto non può cambiare il proprio classificatore.
Contrario: Classificazione dinamica.

Cliente – *Customer*

Persona o organizzazione, interna o esterna a quella che si occupa di produrre il sistema, che si assume le responsabilità finanziarie del sistema stesso. Alcune volte, ma non sempre, si tratta dell'utente finale.

Cliente – *Client*

Classificatore che richiede i servizi esposti da un altro detto fornitore.
Contrario: Fornitore.

Collaborazione – *Collaboration*

Specifica del modo in cui un'operazione o un classificatore, come per esempio un caso d'uso, è realizzato da un insieme di classificatori, associati per mezzo di opportune relazioni, che recitano un determinato ruolo secondo una prestabilita modalità.
Consultare: Interazione.

Collegamento – *Link*
Connessione semantica tra un insieme di oggetti. Si tratta di un'istanza di associazione.
Consultare: Associazione.

Collegamento di fine – *Link end*
Istanza di una fine associazione.
Consultare: Associazione di fine.

Commento – *Comment*
Annotazione attaccata a un elemento o a una collezione. Un commento non ha alcuna semantica associata.
Contrario: Vincolo.

Componente – *Component*
Parte modulare, installabile e modificabile del sistema che incapsula implementazione ed espone un insieme di interfacce. Un componente, tipicamente, è specificato attraverso uno o più classificatori (per esempio classi di implementazione) che lo costituiscono e può essere implementato attraverso diversi manufatti (p.e. file compilati, eseguibili, script, …).
Contrario: Manufatto.

Comportamento – *Behavior*
Effetti osservabili di un'operazione o un evento, inclusi i relativi risultati.

Composizione – *Composition*
Forma particolare e semanticamente più forte di aggregazione che impone due vincoli:
- ogni istanza della parte può partecipare, in ogni istante di tempo, al massimo a una composizione;
- la parte "tutto" (*whole*) deve essere responsabile per la generazione e la distruzione delle parti componenti.

La composizione può essere ricorsiva.
Sinonimo: Aggregazione composta.

Concorrenza – *Concurrency*
Occorrenza di due o più attività durante lo stesso intervallo di tempo. La concorrenza può essere ottenuta attraverso l'esecuzione simultanea o "interlacciata" di due o più *threads* (flussi di esecuzione).
Consultare: Thread.

Condizione di guardia – *Guard condition*
Condizione che deve essere soddisfatta affinché la relativa transizione possa aver luogo.
Consultare: Transizione.

Contenitore – *Container*
1. istanza che esiste per contenere altre istanze e che fornisce operazioni per accedere o iterare il proprio contenuto (per esempio array, liste, set, …)
2. componente che esiste per contenere altri componenti.

Consultare: Componente.

Contesto – *Context*
Vista di un insieme di elementi della modellazione correlati tra loro per un determinato scopo, come specificare un'operazione.

CORBA – *CORBA*
Architettura di un agente comune di richiesta oggetti (Common Object Request Broker Architecture). Specificazione di una architettura a oggetti distribuiti indipendente dal linguaggio di programmazione.

CRC Cards – *CRC Cards*
Classe–Responsabilità–Collaborazione (Class–Responsibility–Collaborators). Tecnica di sviluppo di modelli Object Oriented, inizialmente ideata da Ward Cunningham e Kent Beck al fine di semplificare la modellazione Object Oriented attraverso la definizione formale di cosa una classe dovrebbe fare, ossia le responsabilità, e di quali altre entità sono necessarie per espletare tali responsabilità (collaborazioni).

D

DBMS – *DBMS*
Sistema di gestione delle basi dati (DataBase Management System). Software che gestisce dati opportunamente organizzati, fornendo servizi per il controllo centralizzato, l'indipendenza dei dati, rendere l'accesso più efficiente, garantire l'integrità, il ripristino, il controllo della concorrenza, la privatezza e la sicurezza.

Delega / Delegazione – *Delegation*
Capacità di un oggetto A di inviare un messaggio a un altro oggetto B in risposta della ricezione di un messaggio destinato ad A. La delegazione può essere utilizzata, con opportune cautele e limitazioni, come alternativa all'eredità.
Consultare: Ereditarietà.

Device (Dispositivo) – *Device*
Un tipo di nodo che fornisce proprietà di supporto a un processore. Sebbene abbia le capacità di eseguire programmi *embedded* (*device drivers* necessari per poter fun-

zionare) non può eseguire programmi *general purpose* (normali applicativi). I device vengono introdotti come supporto a particolari processori che eseguono applicazioni general purpose.

Diagramma – *Diagram*

Rappresentazione grafica di una collezione di elementi del modello, tipicamente, riprodotti come grafi connessi da archi (relazioni) e vertici (altri elementi del modello). Lo UML supporta i seguenti diagrammi: dei casi d'uso, di sequenza, di collaborazione, di attività, di stato, delle classi, degli oggetti, dei componenti e di dispiegamento.

Diagramma degli oggetti – *Object diagram*

Diagramma che illustra, in un preciso istante di tempo, un insieme di oggetti opportunamente interconnessi. Può essere considerato un caso particolare di un diagramma delle classi o di uno di collaborazione.
Consultare: Diagramma di collaborazione, Diagramma delle classi.

Diagramma dei componenti – *Component diagram*

Diagramma che mostra l'organizzazione e le dipendenze tra componenti.
Consultare: Componente.

Diagramma delle classi – *Class diagram*

Diagramma che mostra una collezione di elementi dichiarativi del modello, come classi, tipi, loro contenuti e relazioni. Modella la struttura statica di un sistema.

Diagramma di collaborazione – *Collaboration diagram*

Diagramma che mostra interazioni organizzate attorno alla struttura di un modello, utilizzando o classificatori e associazioni oppure istanze e collegamenti. Al contrario dei diagrammi di sequenza, quelli di collaborazione mostrano le relazioni tra le varie istanze. I diagrammi di sequenza e collaborazione mostrano informazioni molto simili (sono isomorfi, è possibile passare direttamente dall'una all'altra forma), ma conferiscono maggiore attenzione a differenti elementi: i diagrammi di sequenza mettono in evidenza il trascorrere del tempo, mentre quelli di collaborazione conferiscono maggiore importanza alla struttura del modello.
Consultare: Diagramma di sequenza.

Diagramma di dispiegamento – *Deployment diagram*

Diagramma che mostra la configurazione a tempo di esecuzione dei nodi di elaborazione e dei componenti corredati dai processi e dagli oggetti in vita su di essi. I componenti sono manifestazioni di unità di codice a tempo di esecuzione.
Consultare: Diagramma dei componenti , Tempo di esecuzione.

Diagramma di sequenza – *Sequence diagram*

Diagramma che mostra l'interazione di oggetti organizzata evidenziando la sequenza temporale. In particolare, mostra lo scambio di messaggi che avviene tra gli oggetti che prendono parte a un'interazione. A differenza del diagramma di collaborazione, quello di sequenza include la sequenza temporale ma non le relazioni tra oggetti. Un diagramma di sequenza può esistere in una forma generica (descrive tutti i possibili scenari) o in una specifica (mostra una particolare istanza di scenario). I diagrammi di sequenza e collaborazione mostrano informazioni molto simili, ma enfatizzando un diverso aspetto (rispettivamente sequenza temporale e relazioni tra oggetti).
Consultare: Diagramma di collaborazione.

Diagramma di stato – *Statechart diagram*

Diagramma che modella una macchina a stati.
Consultare: Macchina a stati.

Diagramma di interazione – *Interaction diagrams*

Termine generico utilizzato per riferirsi a diversi tipi di diagramma (quelli di sequenza e di collaborazione) impiegati per mostrare interazioni tra oggetti.
Consultare: Diagramma di sequenza, Diagramma di collaborazione.

Dipendenza – *Dependency*

Relazione tra due elementi del modello in cui la modifica di uno di essi (quello indipendente) può generare ripercussioni sull'altro (quello dipendente).

Disegno – *Design*

Parte del processo di sviluppo del software il cui obiettivo primario è stabilire come il sistema verrà implementato. Durante la fase di disegno sono prese una serie di decisioni strategiche e tattiche al fine di soddisfare i requisiti, funzionali e non, e i requisiti di qualità del sistema.

Dominio – *Domain*
Area di conoscenza o di attività caratterizzata da un insieme di concetti e terminologie condivisi da addetti ai lavori della specifica area.

E

Elaborazione del modello – *Model elaboration*
Il processo di generazione di un tipo repository a partire da un modello pubblicato. Tale processo include la generazione delle interfacce e dell'implementazione che permettono ai repository di essere istanziati e popolati in funzione e in accordo con il modello elaborato.
Consultare: Repository.

Elemento – *Element*
Costituente atomico di un modello.

Elemento del modello – *Model element [MOF]*
Elemento che costituisce una specifica astrazione del sistema oggetto di modellazione. Nel contesto del MOF (Meta-Object Facility) gli elementi del modello sono considerati metaoggetti.

Elemento derivato – *Derived element*
Elemento del modello che può essere ottenuto a partire da un altro elemento, ma che viene comunque utilizzato o perché contribuisce ad aumentare il livello di chiarezza o per scopi di disegno, anche se non aggiunge alcuna semantica o ulteriori informazioni.

Elemento generalizzabile – *Generalizable element*
Elemento del modello in grado di partecipare a una relazione di generalizzazione nel ruolo di genitore.
Consultare: Generalizzazione.

Elemento parametrizzato – *Parameterized element*
Descrittore di una classe con uno o più parametri cui non è ancora stato attribuito un valore (unbound).
Consultare: Template.

Elemento vista – *View element*
Descrizione testuale e/o proiezione grafica di una collezione di elementi del modello.

Enumerato/Enumerazione – *Enumeration*
Sinonimo: Tipo enumerato.

Ereditarietà – *Inheritance*
Meccanismo che permette a elementi più specifici di incorporare struttura e comportamento definiti in elementi più generali ai quali sono correlati per via di aspetti comportamentali.
Consultare: Generalizzazione.

Ereditarietà dell'interfaccia – *Interface inheritance*
Ereditarietà dell'interfaccia posseduta da un elemento più generale. Non include l'ereditarietà dell'implementazione.
Consultare: Interfaccia.
Contrario: Ereditarietà dell'implementazione.

Ereditarietà dell'implementazione – *Implementation inheritance*
Ereditarietà dell'implementazione di un elemento più generale. Questo tipo di eredità include quella dell'interfaccia.
Contrario: Ereditarietà dell'interfaccia.

Ereditarietà multipla – *Multiple inheritance*
Variazione semantica della generalizzazione nella quale un tipo può avere più di un supertipo (genitore).
Consultare: Supertipo, Tipo.
Contrario: Ereditarietà singola.

Ereditarietà singola – *Single inheritance*
Variazione semantica della relazione di generalizzazione in cui un tipo può avere al massimo un supertipo (genitore).
Contrario: Ereditarietà multipla.

Espressione – *Expression*
Stringa la cui valutazione genera un valore di un tipo specifico. Per esempio la valutazione dell'espressione `(2 * 3 + 7)` genera un valore di tipo numerico.

Espressione booleana – *Boolean expression*
Espressione la cui valutazione restituisce un valore booleano.
Consultare: Booleano.

Espressione temporale – *Time expression*
Espressione la cui valutazione genera un valore di tipo "tempo", assoluto o relativo.

EUP Processo Unificato Esteso – *EUP Enhanced Unified Process*

Versione estesa del processo di sviluppo del software RUP. Prevede che il ciclo di vita del software venga suddiviso in cinque fasi: iniziale, elaborazione, costruzione, transizione e produzione; in più vengono considerati workflow aggiuntivi, quali: gestione delle configurazioni e delle modifiche (*configuration and change management*), gestione del progetto (*project management*), ambientale e gestione infrastrutture.
Consultare: Ciclo di vita del software, RUP, Workflow.

Evento – *Event*

Specificazione di un avvenimento significativa che si manifesta in una ben definita locazione spazio-tempo. Nel contesto dei diagrammi di stato, un evento è un avvenimento in grado di provocare una transizione.

Evento temporale – *Time event*

Evento che rappresenta l'intervallo di tempo trascorso dal momento in cui uno stato concorrente è stato attivato (vi si è transitati).
Consultare: Evento.

Export (Esportazione) – *Export*

Nel contesto dei package, permette di rendere un elemento visibile all'esterno del proprio spazio dei nomi (*namespace*, ambito di denominazione).
Consultare: Package, Spazio dei nomi (namespace), Visibilità.
Contrario: Import (importazione).

Extend – *Extend*

Relazione tra un caso d'uso estendente e uno base, che specifica come il comportamento definito dallo use case estendente è incorporato (sotto il controllo di un'espressione booleana definita nella relazione stessa) nel comportamento del caso d'uso base. Il comportamento è inserito in apposite locazioni definite dai punti di estensione presenti nel caso d'uso base. Quest'ultimo non dipende dall'esecuzione del comportamento del caso d'uso estendente.
Consultare: Caso d'uso, Include, Punto di estensione.

F

Façade – *Façade*

Stereotipo dell'elemento package che contiene esclusivamente riferimenti agli elementi del modello presenti all'interno di un altro package. È utilizzato per fornire una "vista pubblica" di alcuni dei contenuti di un package.

Fase *[processi di sviluppo del software]* – *Phase*

Tempo che intercorre tra due maggiori *milestones*.

Fase di costruzione – *Construction phase*

Terza fase del processo di sviluppo del software durante la quale il software passa dal punto della *baseline* di un'architettura eseguibile al quello in cui è pronto a essere sottoposto alla comunità degli utenti.
Consultare: Baseline.

Fase di elaborazione – *Elaboration phase*

Seconda fase del processo di sviluppo del software durante la quale viene definita l'architettura del sistema.

Fase di principio – *Inception phase*

Prima fase del processo di sviluppo del software dove le prime concettualizzazioni circa lo sviluppo del sistema vengono fatte evolvere al punto da risultare sufficientemente ben fondate per giustificare il passaggio alla fase successiva (elaborazione).

Fase di produzione *[processo di sviluppo del software EUP]* – *Production phase*

Si tratta della quinta fase atta a gestire attività necessarie per mantenere il sistema operativo fino a quando sia disponibile una nuova versione o, addirittura, il sistema venga rimpiazzato da uno completamente nuovo.
Consultare: EUP.

Fase di transizione – *Transition phase*

Quarta fase del processo di sviluppo del software in cui il software è consegnato alla comunità degli utenti.

Figlio – *Child*

Nella relazione di generalizzazione, la specializzazione di un altro elemento detto genitore (*parent*).
Consultare: Sottoclasse, Sottotipo.
Contrario: Genitore (Parent).

Fine associazione – *End association*

Punto di fine di una associazione che connette l'associazione stessa a un classificatore.
Consultare: Associazione, Classificatore.

Firma – *Signature*

Nome e parametri di una caratteristica comportamen-tale. La firma può contemplare un eventuale valore di ritorno.
Consultare: Caratteristica comportamentale.

Fornitore – *Supplier*
Classificatore che fornisce un servizio che può essere invocato da altri detti clienti.
Contrario: Cliente.

Framework (Struttura) – *Framework*
Stereotipo dell'elemento package contenente elementi del modello che realizzano un'architettura riutilizzabile per un intero sistema o per sue parti. I framework, tipicamente, includono classi, interfacce, pattern e/o template.
Quando i framework sono specializzati per uno specifico dominio, spesso ci si riferisce ad essi con il nome di framework applicativi.
Consultare: Package, Pattern.

G

Generalizzazione – *Generalization*
Relazione tassonomica tra un elemento più generale e uno più specifico. Quest'ultimo è completamente consistente con l'elemento più generale e contiene informazioni supplementari. Ogni istanza dell'elemento più specifico può essere utilizzata in ogni posto in cui è permesso l'utilizzo dell'elemento più generale.
Consultare: Ereditarietà.

Genitore – *Parent*
Generalizzazione di un altro elemento, detto figlio.
Consultare: Superclasse, Supertipo.
Opposto: Figlio.

Gerarchia di contenimento – *Containment hierarchy*
Consiste in elementi del modello corredati dalle relative relazioni di contenimento. Una gerarchia di contenimento costituisce un grafo.

Grafo di attività – *Activity graph*
Caso particolare di una macchina a stati (*state machine*) utilizzata per modellare processi che coinvolgono uno o più classificatori.
Consultare: Macchina a stati.
Contrario: Diagramma di stato (statechart diagram).

Guidato dai casi d'uso – *Use case-driven*
Nel contesto dei processi di sviluppo del software, approccio in cui i casi d'uso sono utilizzati come manufatto primario per stabilire il comportamento desiderato del sistema e per comunicare tale comportamento tra i vari soggetti coinvolti nello sviluppo del sistema stesso. Lo stesso approccio comporta che il modello dei casi d'uso diventi l'input primario per le restanti fasi di analisi, disegno, implementazione e test del sistema includendo la creazione, la verifica e la validazione dell'architettura del sistema.

Guidato dai fattori di rischio – *Risk-driven*
Nel contesto dei processi di sviluppo del software, strategia che consiste nel focalizzare l'attenzione, in ogni nuova release del sistema, sull'individuazione e sulla riduzione dei rischi contingenti più significativi e pressanti che possono minare il buon successo del progetto.

I

ICONIX – *ICONIX*
Esempio di processo di sviluppo del software basato sull'approccio use case-driven (guidato dai casi d'uso)
Consultare: Guidato dai casi d'uso.

Implementazione – *Implementation*
Definizione del modo in cui qualcosa viene costruito o elaborato. Per esempio, una classe è l'implementazione di un tipo, un metodo è l'implementazione di un'operazione, e così via.
Contrario: Specificazione.

Import (Importazione) – *Import*
Dipendenza tra package che evidenzia i package le cui classi possono essere referenziate all'interno di un altro, e/o ricorsivamente, in tutti i package in esso contenuti.
Contrario: Export (Esportazione).

Include – *Include*
Relazione tra un caso d'uso base e uno incluso, che specifica il modo in cui il comportamento definito dal primo contiene il comportamento specificato nel caso d'uso incluso. Il comportamento di quest'ultimo è inserito in apposite locazioni definite nel caso d'uso in base. Quest'ultimo pertanto dipende dall'esecuzione dello use case incluso, ma non dalla sua struttura (attributi e operazioni).
Consultare: Caso d'uso, Extend.

Ingegnerizzazione inversa *[processi di sviluppo del software]* – *Reverse engineering*
La trasformazione del codice in un modello attraverso un'opportuna corrispondenza dallo specifico linguaggio di programmazione utilizzato agli elementi del modello.
Consultare: Ingegnerizzazione diretta.

Ingegnerizzazione diretta – *Forward engineering*

Nel contesto dei processi di sviluppo del software indica la trasformazione di un modello nel rispettivo codice attraverso un'opportuna corrispondenza con uno specifico linguaggio di programmazione.
Contrario: Ingegnerizzazione inversa.

Innescare – *Fire*

Eseguire una transizione di stato.
Consultare: Transizione.

Interazione – *Interaction*

Specificazione di come determinati stimoli vengono scambiati tra particolari istanze al fine di eseguire un compito prestabilito. L'interazione è definita nel contesto di una collaborazione.
Consultare: Collaborazione.

Interfaccia – *Interface*

Insieme di operazioni, identificate da un nome, utilizzato per caratterizzare il comportamento di un elemento.
Consultare: Operazione.
Contrario: Tipo.

Invio di un messaggio – *Send [message]*

Passaggio di uno stimolo da un'istanza mittente a una destinataria.
Consultare: Sender (Mittente), Destinatario.

Istanza – *Instance*

Entità, identificata da un'identità univoca, alla quale può essere applicato un insieme ben definito di operazioni, e che possiede uno stato in grado di memorizzare gli effetti dell'esecuzione di queste operazioni.
Consultare: Oggetto.

Istanze di casi d'uso – *Use case instances*

Esecuzione di una sequenza di azioni specificate in un caso d'uso.
Consultare: Caso d'uso.

Iterativo *[contesto dei processi di sviluppo del software]* – *Iterative*

Indica la strategia che prevede la gestione di una successione di release via via più complete.

Iterazione – *Iteration*

Insieme distinto di attività eseguite in accordo a uno specifico piano e criteri di valutazione il cui risultato consiste in un nuova release del sistema.

L

Linea di vita di un oggetto – *object lifeline*

Linea di un diagramma di sequenza (tipicamente tratteggiata) ortogonale ad un oggetto che rappresenta l'esistenza dello stesso durante uno specifico intervallo di tempo.
Consultare: Diagramma di sequenza, oggetto.

Localizzazione del controllo – *Focus of control*

Simbolo appartenente ai diagrammi di sequenza che mostra l'intervallo di tempo durante il quale l'efferente oggetto esegue un'azione, sia direttamente, sia attraverso una procedura subordinata.
Consultare: Diagramma di sequenza

M

Macchina a stati – *State machine*

Comportamento che specifica le sequenze degli stati, corredati dalle risposte e dalle azioni intraprese, che un oggetto o un'interazione deve percorrere durante la propria esistenza in risposta a determinati eventi.

Manufatto – *Artifact*

Rappresentazione fisica di un'unità di informazione utilizzata dal processo di sviluppo del software. Esempi di manufatti sono i modelli, i file sorgenti, gli script, i file eseguibili, e così via. Un manufatto potrebbe costituire l'implementazione di un componente eseguibile.
Sinonimo: Prodotto.

Meccanismo – *Mechanism*

Soluzione comune per un problema o requisito ricorrente. Esempi sono i meccanismi di disegno che provvedono la persistenza degli oggetti o gli strumenti di distribuzione dei modelli di disegno.

Messaggio – *Message*

Specificazione di un conveniente scambio di informazione tra istanze, con l'aspettativa che abbia luogo una determinata attività. Un messaggio può specificare l'invio di un segnale o l'invocazione di una operazione.

Metaclasse – *Metaclass*

Classe le cui istanze sono a loro volta classi. Le metaclassi tipicamente sono utilizzate per costruire metamodelli.
Consultare: Metamodello.

Meta-metamodello – *Meta-metamodel*
Modello che definisce il linguaggio necessario per esprimere metamodelli. La relazione tra il meta-metamodello e un metamodello è analoga alla relazione che intercorre tra un metamodello e un modello.
Consultare: Metamodello.

Metamodello – *Metamodel*
Modello che definisce il linguaggio necessario per esprimere un modello.

Metaoggetto – *Metaobject*
Termine generico utilizzato per riferirsi a tutte le metaentità di un linguaggio di modellazione. Esempi di metaoggetto sono metatipi, metaclassi, metaattributi e metaassociazioni.

Metodo – *Method*
Implementazione di un'operazione. Specifica l'algoritmo o la procedura associata a una operazione.
Consultare: Operazione.

Modello – *Model*
Astrazione di un sistema fisico realizzata per scopi specifici.
Consultare: Sistema fisico.

Modello *[Contesto della specifica MOF (Meta-Object Facility)]* – *Model*
Descrive un meta-metamodello, il quale spesso, per questioni di brevità, è riferito semplicemente con il nome di modello.
Consultare: Meta-metamodello.

Modello a oggetti del dominio – *Business object model*
Modello a oggetti (insieme di diagrammi delle classi) che descrive la realizzazione dei casi d'uso business.

Modello dei casi d'uso – *Use case model*
Modello che descrive i requisiti funzionali di un sistema in termini di casi d'uso.
Consultare: Caso d'uso.

Modello di definizione *[MOF]* – *Defining model [MOF]*
Modello sul quale è basato il repository. Un qualsiasi numero di repository può far riferimento al medesimo modello di definizione.
Consultare: Repository.

Modello libreria – *Model library*
Stereotipo dell'elemento package che contiene elementi del modello con l'obiettivo di riutilizzarli in altri package. Un modello di libreria differisce da un profilo in quanto il primo non estende il metamodello attraverso l'utilizzo dei meccanismi propri di estensione come gli stereotipi, i valori etichettati e vincoli. Un modello di libreria è analogo al concetto di libreria di classi proprio dei linguaggi di programmazione.
Consultare: Package, Profilo.

Modello pubblicato [MOF] – *Published model*
Modello "congelato" e reso disponibile per istanziare repository e per il supporto nella definizione di altri modelli.Gli elementi di un modello "congelato" non possono essere modificati.
Consultare: Repository.

Modificatore di accesso – *Access modifier*
Parola chiave del linguaggio utilizzata per modificare l'accesso a classi, metodi e attributi.
Consultare: Visibilità.

Modulo – *Module*
Unità di software di memorizzazione e manipolazione. Sono moduli quelli di codice sorgente, di codice binario, di codice eseguibile, ecc.
Consultare: Componente.

MOF Supporto per metaoggetti – MOF *Meta-Object Facility*
Specificazione che definisce un insieme di interfacce CORBA (CORBA IDL) che possono essere utilizzate per definire e manipolare un insieme di metamodelli interoperabili (lo UML ne è un esempio) e i relativi modelli.

Molteplicità – *Multiplicity*
Specifica di un intervallo di cardinalità consentite che un insieme può assumere. La molteplicità può essere specificata per un ruolo di un'associazione, per le parti di una composizione, per ripetizioni e così via. Una molteplicità è un sottoinsieme, eventualmente infinito, di numeri naturali (interi non negativi).
Contrario: Cardinalità.

Monovalorizzato – *Single-valued [MOF]*
Elemento del modello con molteplicità definita il cui valore superiore dell'attributo di `MultiplicityType::` è impostato a un `1`. Il termine *single-valued* pertanto non

si riferisce al numero di valori che un attributo, un parametro, ecc. può assumere in ogni istante di tempo: un attributo single-valued potrebbe non aver alcun valore (per esempio molteplicità uguale a zero).
Contrario: Plurivalorizzato.

N

Nodo – *Node*
Classificatore che rappresenta, in fase di esecuzione, una risorsa tipicamente dotata almeno di memoria, alla quale frequentemente abbina capacità elaborative, sede di componenti e oggetti eseguibili.

Nome – *Name*
Stringa utilizzata per identificare un elemento del modello.

Non interpretato – *Uninterpreted*
Segnaposto (*placeholder*) per uno o più tipi la cui implementazione non è specificata dallo UML. Ogni valore non interpretato ha una corrispondente stringa che ne fornisce la rappresentazione.

O

Oggetto –*object*
Entità ben delimitata, dotata di nome che incapsula stato e comportamento. Lo stato, in un determinato istante di tempo, è rappresentato dal valore assunto da tutti gli attributi e dalle relazioni instaurate con gli altri oggetti. Il comportamento invece è rappresentato da operazioni, metodi e macchine di stato. Un oggetto è istanza di una specifica classe.
Consultare: Classe, Istanza.

Oggetto attivo – *Active object*
Oggetto istanza di una classe attiva che possiede il thread (flusso di esecuzione) e che può avviare un'attività di controllo.
Consultare: Classe attiva, Thread (flusso).

Oggetto persistente – *Persistent object*
Un oggetto che continua a esistere dopo il termine del processo o del flusso di esecuzione (*thread*) che lo ha creato.

Oggetto transiente – *Transient object*
Oggetto che esiste esclusivamente durante l'esecuzione del processo o del flusso di esecuzione (thread) che lo ha creato.
Consultare: Flusso (thread).

OMG Object Management Group
Si tratta del gruppo no profit a cui aderisce la quasi totalità dei vendor appartenenti alla comunità Object Oriented. Il fine di tale organizzazione è promuovere standard *de facto* relativi al mondo OO atti a incoraggiare lo sviluppo di tale paradigma. Attualmente le relative responsabilità includono, tra l'altro, la gestione dell'evoluzione dello UML e dell'architettura CORBA.

Operazione – *operation*
Servizio che può essere richiesta ad un determinato oggetto la cui esecuzione ne influenza il comportamento. Un'operazione dispone di una precisa firma che limita l'insieme dei parametri attuali ammissibili.
Consultare: Firma, Parametri attuali.

P

Package – *Package*
Meccanismo generalizzato utilizzato per organizzare elementi in gruppi. È possibile dar luogo a package annidati in altri package.

Parametro –*parameter*
Specifica di una variabile che può essere modificata, fornita o restituita in ritorno. Un parametro può includere un nome, tipo e direzione. I parametri sono utilizzati per operazioni, messaggi ed eventi.
Sinonimo: Parametro formale.
Opposto: Argomento.

Parametro formale – *Formal parameter*
Sinonimo: Parametro

Parametro attuale – *Actual parameter*
Sinonimo: Argomento.

Partecipare – *Participate*
Connessione di un elemento ad una relazione. Per esempio, una classe partecipa in un'associazione, un attore partecipa in un caso d'uso, ecc.

Partizione – *Partition*
1. grafi di attività: porzione di un grafo di attività che organizza le responsabilità delle azioni. *Consultare*: Swimlane.
2. architettura: insieme di classificatori correlati presenti allo stesso livello di astrazione appartenenti a diversi strati dell'architettura. Una

partizione rappresenta una frammentazione verticale trasversale all'architettura, mentre uno strato ne à una porzione orizzontale. *Contrario*: Strato.

Pattern – *Pattern*

Template di collaborazione. Un pattern di disegno fornisce uno schema per perfezionare i sottosistemi o i componenti di un sistema software, e/o le relazioni tra di essi. Descrive una struttura ricorrente di componenti comunicanti che risolvono un problema generale di disegno all'interno di uno specifico contesto.

Pattern architetturali – *Architectural patterns*

Pattern che definiscono determinate strutture o comportamenti, tipicamente utilizzati per la vista architetturale o per uno specifico modello. Esempi di pattern architetturali sono three-tier, multi-tier, client-server, e altri, ognuno dei quali definisce una specifica struttura del modello di dispiegamento e fornisce le linee guida sul modo in cui i componenti devono essere allocati nei vari nodi.

Piano di iterazione – *Iteration plan*

Piano dettagliato per una specifica iterazione. Piano che dichiara il costo previsto in termini di tempo e di impiego di risorse e i risultati attesi in termini di manufatti per una specifica iterazione. Un piano di iterazione deve anche specificare chi dovrebbe fare cosa nel corso dell'iterazione e in quale ordine. Ciò è ottenuto allocando le singole persone ai "lavoratori" (sorta di segnaposto) e descrivendo in dettaglio il workflow dell'iterazione.
Consultare: Iterazione, Manufatto, Worker (Lavoratore), Workflow.

Piano di test – *Test plan*

Piano che specifica le strategie di test, le risorse e la relativa schedulazione.

Plurivalorizzato – *Multi-valued* *[MOF]*

Elemento del modello con molteplicità definita il cui valore superiore dell'attributo `MultiplicityType::` è impostato a un numero maggiore di `1`. Il termine *multi-valued* pertanto non si riferisce al numero di valori che un attributo, un parametro, ecc. può memorizzare in ogni istante di tempo.
Contrario: Monovalorizzato.

Portabilità – *Portability*

Grado in cui un sistema eseguibile in uno specifico ambiente di esecuzione può essere facilmente spostato in un altro ambiente di esecuzione mantenendo la proprietà di essere eseguibile.

Postcondizione – *Post-condition*

Vincolo che deve essere soddisfatto al termine dell'esecuzione di un'operazione.

Precondizione – *Pre-condition*

Vincolo che deve essere soddisfatto all'atto dell'invocazione di un'operazione.

Processo – *Process*

Unità di esecuzione pesante che può essere eseguita in concorrenza con altri thread in un sistema operativo. Si contrappone al concetto di thread il quale include processi pesanti e leggeri. Se necessario è possibile distinguere i due concetti attraverso opportuni stereotipi.
Spesso indica l'esecuzione di un algoritmo o altrimenti gestione di un'attività dinamica.
Contrario: Thread (flusso di esecuzione).

Processo *[processo di sviluppo del software]* – *Process*

Attività, coordinamento e linee guida da seguire per costruire un sistema.

Procedura di test – *Test procedure*

Specifica sul modo in cui eseguire uno o più casi di test.
Consultare: caso di test.

Processo business – *Business process*

Insieme di attività necessarie per produrre come risultato un valore compreso e misurabile per un cliente individuale di un particolare business.

Processo di sviluppo – *Development process*

Insieme di passi ordinati eseguiti per uno scopo prestabilito durante lo sviluppo del software, come costruzione e implementazione di modelli.

Processo di sviluppo del software unificato – *Unified Software Development Process*

Processo di sviluppo del software fondato sullo UML che integra gli approcci iterativo e incrementale, centrato sull'architettura, guidato dai casi d'uso e dai fattori di rischio. Si tratta di un processo organizzato su quattro fasi (inizio, elaborazione, costruzione e transizione) e su cinque workflow principali (cattura dei requisiti, analisi, disegno, implementazione e test). Processo descritto in termini di un modello di business, il quale è struttu-

rato in termini di tre blocchi primitivi (lavoratori, attività e manufatti).

Processor (Processore) – *Processor*

Tipo di nodo che possiede le capacità di eseguire uno o più processi. Queste includono capacità di esecuzione, memoria, dispositivi di input/output e così via.
Consultare: Device, Nodo, Processo.

Profilo – *Profile*

Stereotipo di package che contiene elementi del modello "customizzati" (specializzati) per uno specifico dominio o scopo utilizzando i meccanismi di estensione propri dello UML: stereotipi, valori etichettati e vincoli. Un profilo può eventualmente specificare sia modelli di libreria dal quale dipende, sia il sottoinsieme del metamodello che estende.
Consultare: Modello di libreria, Package.

Proiezione – *Projection*

Corrispondenza (*mapping*) tra un insieme e un suo sottoinsieme.

Proiezione vista – *View projection*

Proiezione di elementi del modello in un elemento vista. La proiezione vista fornisce una locazione e uno stile per ogni elemento vista.

Proprietà – *Property*

Valore identificato da un nome denotante una specifica caratteristica di un elemento. Le proprietà hanno un impatto semantico. Alcune proprietà sono predefinite nello UML, altre possono essere definite dall'utente.
Consultare: Valore etichettato.

Proprietà comportamentali – *Behavioral features*

Proprietà dinamica di un elemento del modello, come un'operazione o un metodo.

Pseudostato – Pseudo-state

Vertice (nodo) in una macchina a stati che ha la forma dello stato, ma che non possiede comportamento da stato. Esempi di pseudostati sono quello iniziale e quello storico.
Consultare: Stato iniziale, Stato storico.

Punto di variazione semantica – *Semantic variation point*

Punto di variazione nella semantica di un metamodello. Si tratta di un intenzionale grado di libertà utilizzato per semplificare l'interpretazione della semantica del metamodello.

Q

Qualificatore – *Qualifier*

Attributo o *t*-upla di attributi appartenenti ad una specifica associazione i cui valori partizionano l'insieme degli oggetti relazionati ad un altro attraverso un'associazione.

R

Raffinamento – *Refinement*

Relazione che rappresenta una più completa specifica di un qualcosa già definito con un determinato livello di astrazione. Per esempio un modello di disegno è un raffinamento del modello di analisi.

Regola business – *Business rule*

Si tratta di principi, policy, leggi, dettagliati algoritmi di calcolo, relazioni tra oggetti, ecc. che influenzano direttamente il dominio del problema che il sistema dovrà, in qualche misura, automatizzare. Pertanto, rappresentano informazioni che, in ultima analisi, dovranno confluire nell'implementazione del sistema.

Relazione – *Relationship*

Connessione semantica tra elementi del modello. Esempi di relazioni sono l'associazione, la generalizzazione, ecc.

Repository – *Repository*

Meccanismo per la memorizzazione dei modelli ad oggetti, interfacce e implementazioni.

Requisito – *Requirement*

Condizione o capacità cui un sistema deve conformarsi.

Requisito funzionale – *Functional requirement*

Requisito che specifica un'azione che il sistema deve essere in grado di eseguire senza prendere in considerazione vincoli di carattere fisico (provenienti principalmente dall'architettura). In altre parole un requisito che specifica gli output che il sistema deve produrre a fronte di determinati input.

Requisito non funzionale – *Non functional requirement*

Requisito che specifica proprietà richieste al sistema, come vincoli ambientali e di sviluppo, prestazioni, dipendenze dalla piattaforma, di manutenibilità, estendibilità, sicurezza e affidabilità. Requisito che sancisce vincoli di carattere fisico relativi ai requisiti funzionali.
Contrario: Requisito funzionale.

Requisito prestazionale – *Performance requirement*

Requisito che impone condizioni comportamentali a requisiti funzionali, come velocità, numero di lavori eseguiti nell'unità di tempo (*throughput*), tempo di risposta e utilizzo della memoria.
Consultare: Requisito non funzionale.

Responsabilità – *Responsability*

Contratto o obbligazione di un classificatore.
Consultare: Classificatore.

Ricevente [oggetto] – *Receiver [object]*

Oggetto che tratta lo uno stimolo fornito da un altro oggetto detto mittente.
Opposto: Mittente.

Ricevere [messaggio] – *Receive [message]*

Trattamento di uno stimolo fornito da un oggetto mittente.
Consultare: Oggetto mittente.

Riferimento – *Reference*

1. Denotazione di un elemento del modello.
2. Slot (spazio) di un classificatore, identificato da un nome, che facilita la navigazione verso altri classificatori.
Consultare: Classificatore.

Rischio – *Risk*

Variabile del progetto in grado di mettere in pericolo o addirittura annullare il successo dell'intero processo. I rischi possono essere o meno di natura tecnica e possono far sì che il progetto vada incontro a eventi indesiderati come ritardi rispetto alla pianificazione, incremento dei costi e improvvise cancellazioni.

Riuso – *Reuse*

Utilizzo di un manufatto (o opportune porzioni) preesistente.

Ruolo – *Role*

Comportamento specifico, dotato di nome, assunto da un'entità che prende parte a un determinato contesto. Un ruolo può essere statico (come nel caso dell'associazione di fine) o dinamico (come nel ruolo della collaborazione).

RUP Processo unificato della Rational – *RUP Rational Unified Process*

Processo di sviluppo del software fornito dalla Rational, evoluzione dello Unified Software Development Process.
Consultare: Processo di sviluppo del software unificato.

S

Scenario – *Scenario*

Sequenza specifica di azioni che illustrano un determinato comportamento. Uno scenario può essere utilizzato per mostrare una particolare interazione o l'esecuzione di una specifica istanza di un caso d'uso.
Consultare: Interazione.

Schema *[MOF]* – *Schema [MOF]*

Nel contesto del Meta-Object Facility (MOF), uno schema è analogo a un package contenente elementi del modello; in altre parole uno schema corrisponde a un package del MOF.
Contrario: Metamodello, Package.

Sender (Mittente) *[oggetto]* – Sender *[object]*

Oggetto che passa lo stimolo (invia un messaggio) a un oggetto destinatario.
Contrario: Destinatario, Messaggio.

Sequenza di azione – *Action sequence*

Espressione che si risolve in una sequenza di azioni.
Consultare: Azione.

Segnale – *Signal*

Specifica della comunicazione di un stimolo asincrono tra istanze. I segnali possono essere dotati di parametri.

Sistema – *System*

Sottosistema di primo livello (*top level*) in un modello.
Consultare: Top level.
Contrario: Sistema fisico.

Sistema fisico – *Physical system*

1. Soggetto di un modello.

2. Collezione di unità fisiche interconnesse, le quali possono includere software, hardware e persone, organizzate al fine di ottenere uno specifico risultato. Un sistema fisico si presta ad essere descritto per mezzo di uno o più modelli, realizzati secondo differenti punti di vista.
Contrario: Sistema.

Sistema legacy – *Legacy system*
Sistema preesistente che il nuovo progetto deve "ereditare". Tipicamente si tratta di sistemi datati, realizzati con tecnologie ormai obsolete, che comunque deve essere incorporato o riutilizzato, totalmente o in parte, dal nuovo sistema da realizzare.

Sottoclasse – *Subclass*
Nella relazione di generalizzazione, si tratta della specializzazione di un'altra classe definita superclasse o genitore.
Consultare: Generalizzazione
Contrario: Superclasse.

Sottomacchina a stati – *Submachine state*
Stato di una macchina a stati equivalente a uno stato composto il cui comportamento però è definito per mezzo di un'altra macchina a stati.

Sottopackage – *Subpackage*
Package contenuto in un altro package.
Consultare: Package.

Sottosistema – *Subsystem*
Gruppo di elementi del modello che rappresentano un'unità comportamentale di un sistema fisico. Un sottosistema espone interfacce e possiede delle operazioni. Inoltre, gli elementi del modello di un sottosistema possono essere partizionati in elementi di specifica e elementi di realizzazione.
Consultare: Package, Sistema fisico.

Sottostato – *Substate*
Stato parte di uno stato composto.
Consultare: Stato concorrente, Stato disgiunto.

Sottostato concorrente – *Concurrent substate*
Sottostato che può essere contenuto simultaneamente con altri sottostati nel medesimo stato composto.
Consultare: Stato composto.
Contrario: Sottostato disgiunto.

Sottostato disgiunto – *Disjoint substate*
Sottostato che non può essere contenuto simultaneamente con altri nel medesimo stato composto.
Consultare: Stato composto.
Contrario: Sottostato concorrente.

Sottotipo – *Subtype*
In una relazione di generalizzazione, rappresenta la specializzazione di un altro tipo: il supertipo.
Consultare: Generalizzazione.
Contrario: Supertipo.

Spazio dei nomi – *Namespace*
Parte del modello in cui i nomi possono essere definiti e utilizzati. All'interno di uno spazio dei nomi, ogni nome deve essere univoco.
Consultare: Nome.

Specificazione – *Specification*
Descrizione dichiarativa di che cosa un'entità è o del suo comportamento.
Contrario: Implementazione.

Stato – *State*
Condizione o situazione durante il ciclo di vita di un oggetto durante la quale l'oggetto stesso soddisfa particolari condizioni, esegue determinate attività o attende specifici eventi.

Stato composto – *Composite state*
Stato che può essere costituito o da altri sottostati concorrenti (ortogonalmente) oppure da sottostati sequenziali (disgiunti).
Consultare: Sottostato.

Stato di azione – *Action state*
Uno stato che rappresenta l'esecuzione di un'azione atomica: si tratta tipicamente dell'invocazione di una specifica operazione.
Consultare: Stato.

Stato flusso di oggetto – *object flow state*
Stato in un grafo di attività che rappresenta il passaggio di un oggetto dall'output di azioni appartenenti ad uno specifico stato, alle azioni di input di un altro stato.
Consultare: Azione, Oggetto.

Stato di sottoattività – *Subactivity state*
Stato di un grafo di attività rappresentante l'esecuzione di una sequenza non atomica di passi avente una qualche durata.

Stato finale – *Final state*
Caso particolare di stato specificante che lo stato composto che lo contiene o l'intera macchina a stati è conclusa.
Consultare: Stato.

Stato iniziale – *Initial state*
Tipo particolare di stato che specifica la sorgente di una singola transizione o lo stato di default di uno stato composto.
Consultare: Stato composto, Transizione.

Stato di sincronizzazione – *Synch state*
Vertice in una macchina a stati utilizzato per sincronizzare le aree di concorrenza in una macchina a stati.
Consultare: Vertice.

Stereotipo – *Stereotype*
Nuovo tipo di elemento del modello in grado di estendere la semantica del metamodello. Gli stereotipi devono essere basati su specifici tipi o classi appartenenti al metamodello, dei quali possono estendere la semantica, ma non la struttura. Alcuni stereotipi sono predefiniti nello UML mentre altri possono essere definiti dall'utente. Gli stereotipi rappresentano uno dei tre meccanismi di estensione previsti dallo UML.
Consultare: Valore etichettato, Vincolo.

Stimolo – *Stimulus*
Passaggio di informazioni da un'istanza ad un'altra, che può verificarsi inviando un segnale o invocando un'operazione. Un evento consiste nella ricezione di un segnale.
Consultare: Evento, Messaggio.

Strato – *Layer*
Organizzazione di un insieme di classificatori o di package che si trovano allo stesso livello di astrazione. Uno strato rappresenta una sezione orizzontale di un'architettura, mentre una partizione ne rappresenta una sezione verticale.
Contrario: Partizione.

Stringa – *String*
Sequenza di caratteri. I dettagli della rappresentazione delle stringhe dipendono dalla particolare implementazione considerata, la quale può includere insiemi di caratteri internazionali e grafici.

Superclasse – *Superclass*
In una relazione di generalizzazione, rappresenta la generalizzazione di un'altra classe: la sottoclasse o figlia (*child*).
Consultare: Generalizzazione.
Contrario: Sottoclasse.

Supertipo – *Supertype*
In una relazione di generalizzazione, rappresenta la generalizzazione di un altro tipo: il sottotipo.
Consultare: Generalizzazione.
Contrario: Sottotipo.

Swimlane (Corsia di nuoto) – *Swimlane*
Partizione di un diagramma di attività utilizzata al fine di organizzare le azioni in base alle relative responsabilità. Le swimlane tipicamente corrispondono a unità organizzative di un modello business.
Consultare: Partizione.

T

Template – *Template*
Sinonimo: Elemento parametrizzato.

Tempo di analisi – *Analysis time*
Si riferisce a qualcosa che avviene durante la fase di analisi del processo di sviluppo del software.
Consultare: Tempo di disegno, Tempo di modellazione.

Tempo di compilazione – *Compile time*
Si riferisce a un'azione che avviene durante la compilazione di un modulo software.
Consultare: Tempo di esecuzione, Tempo di modellazione.

Tempo di disegno – *Design time*
Si riferisce a qualcosa che si verifica durante la fase di disegno del processo di sviluppo del software.
Consultare: Disegno, Tempo di modellazione.
Contrario: Tempo di analisi.

Tempo di esecuzione – *Run time*
Intervallo di tempo durante il quale un programma per computer è in esecuzione.
Contrario: Tempo di modellazione.

Tempo di modellazione – *Modeling time*
Si riferisce a qualcosa che avviene durante una fase di modellazione del processo di sviluppo del software. Include tempo di analisi e di disegno.
Quando si parla di sistemi implementati utilizzando il paradigma Object Oriented, spesso risulta molto importante distinguere chiaramente problemi relativi al tempo di modellazione da quelli attinenti il tempo di esecuzione.
Consultare: Tempo di disegno, Tempo di analisi.
Contrario: Tempo di esecuzione.

Test – *Test*
Workflow il cui obiettivo è verificare che un particolare manufatto sia conforme ai relativi requisiti. Il test più importante è quello che consiste nel verificare le caratteristiche e la rispondenza ai requisiti di ogni build del sistema e in particolare la versione consegnata all'utente.
Consultare: Build.

Thread (Flusso di esecuzione) – *Thread* (of control)
Singolo percorso di esecuzione all'interno di un programma o di un modello o di qualsiasi altra rappresentazione del flusso di esecuzione. Si tratta anche di uno stereotipo (*thread*) utilizzato per la realizzazione di un oggetto attivo come un processo leggero.
Consultare: Processo.

Tipo – *Type*
Stereotipo dell'elemento classe che specifica un dominio di oggetti corredati dalle operazioni ad essi applicabili, senza definirne l'implementazione fisica. Un tipo può non contenere alcun metodo, mantenere il proprio flusso di esecuzione (*thread*), e/o essere annidato. Comunque può possedere attributi e instaurare associazioni con altri tipi. Sebbene un oggetto possa prevedere al più una classe di implementazione, potrebbe essere conforme a molti tipi diversi.
Consultare: Classe implementazione.
Contrario: Interfaccia.

Tipo di dato – *Datatype*
Descrittore di un insieme di valori privi di identità e di un insieme di operazioni, a questi applicabili, che non prevedono effetti collaterali. I tipi di dato includono quelli predefiniti e quelli definiti dall'utente. Alla prima categoria appartengono numeri, stringhe e tempo. Alla categoria dei tipi di dato definiti dall'utente appartengono i tipi enumerati.
Consultare: Tipo enumerato.

Tipo enumerato – *Enumeration*
Lista di valori identificati da un nome, utilizzata come dominio dei valori assegnabili a un particolare tipo di attributo. Per esempio, `Visibilità = {public, protected, private, package}`.

Tipo espressione – *Type expression*
Tipo risultato della valutazione di un'espressione.
Consultare: Tipo.

Tipo primitivo – *Primitive type*
Tipo di dato primitivo e predefinito senza sottostruttura. Alcuni esempi di tipo di dato primitivo: intero, reale, …

Top level (Livello superiore) – *Top level*
Stereotipo di package che denota il package più elevato (radice) in una gerarchia di contenimento. Questo stereotipo (*topLevel*) definisce i limiti esterni per la ricerca dei nomi in uno specifico spazio dei nomi.
Consultare: Package.

Traccia – *Trace*
Stereotipo della relazione di dipendenza indicante una relazione di storicità o di processo tra due elementi che rappresentano lo stesso concetto senza indicare le regole specifiche per derivare l'uno dall'altro.
Consultare: Dipendenza, Processo.

Transizione – *Transition*
Relazione tra due stati indicante che un oggetto nel primo stato, in risposta della ricezione di un determinato evento e risultando soddisfatte precise condizioni, è in grado di transitare nel secondo stato, eseguendo specifiche azioni. Quando avviene il cambiamento di stato, si dice che la transizione è innescata (*fire*).
Consultare: Innescare (Fire), Stato.

Transizione interna – *Internal transition*
Transizione prodotta come risultato di un evento, che non genera il cambiamento di stato di un oggetto.

U

UML Linguaggio di modellazione unificato – UML Unified Modeling Language
Specifica che definisce un linguaggio grafico per visualizzare, specificare, costruire, e documentare manufatti di un sistema a oggetti distribuito. La specifica include la definizione formale di un metamodello comune di analisi e disegno (OA&D), una notazione gra-

fica e interfacce CORBA IDL che rendono possibile l'interscambio di modelli tra diversi tool e tipi di repository per metadata.

Unità di distribuzione – *Distribution unit*

Insieme di oggetti o componenti allocati in gruppo (unitariamente) ad un processo o processore. Un'unità di distribuzione può essere rappresentata come una composizione o aggregazione a tempo di esecuzione.
Consultare: Aggregazione, Composizione, Tempo di esecuzione.

Usage (Utilizzo) – *Usage*

Dipendenza in cui un elemento (cliente) richiede la presenza di un altro (fornitore) per il suo corretto funzionamento e la sua implementazione.

Utilità – *Utility*

Stereotipo che raggruppa variabili globali e procedure nella forma della dichiarazione di una classe. Gli attributi e le operazioni dell'utilità diventano, rispettivamente, variabili e procedure globali. Un'utilità non è un costrutto di modellazione fondamentale, ma un espediente utile alla programmazione.

V

Valore – *Value*

Elemento di uno specifico tipo di dominio.
Consultare: Tipo.

Valore etichettato – *Tagged value*

Definizione esplicita di una proprietà attraverso l'accoppiamento del nome a uno specifico valore. Nei valori etichettati, tale nome è denominato etichetta. Specifici valori etichettati sono predefiniti nello UML e altri possono essere definiti dall'utente. I valori etichettati rappresentano uno dei tre meccanismi standard previsti dallo UML per estenderne la semantica.
Consultare: Stereotipo, Vincolo.

Versione – *Release*

Insieme di manufatti relativamente completo e consistente, che possibilmente includa anche il relativo build.

Vertice – *Vertex*

Sorgente o destinazione di una transazione in una macchina a stati. Un vertice può essere sia uno stato, sia un pseudostato.
Consultare: Pseudostato, Stato.

Vincolo – *Constraint*

Condizione semantica o restrizione. Alcuni vincoli sono predefiniti in UML, altri possono essere definiti dagli utenti. I vincoli costituiscono uno dei tre meccanismi di estensione previsti dallo UML.
Consultare: Valore etichettato, Stereotipo.

Visibilità – *Visibility*

Tipo enumerato i cui valori (`public`, `private`, `protected` e `package`) denotano come l'elemento del modello al quale è riferito può essere visto al di fuori del proprio spazio di denominazione (namespace).
Consultare: Spazio di denominazione.

Vista – *View*

Proiezione di un modello, realizzata da una particolare prospettiva al fine di enfatizzare specifici aspetti e di omettere altre entità che non sono rilevanti per i fini della proiezione stessa.

Vista architetturale – *Architectural view*

Proiezione della struttura e comportamento di uno specifico modello di un sistema, focalizzato sugli aspetti architetturali significativi dello stesso modello.

W

Worker (Lavoratore) – *Worker*

Nei processi di sviluppo del software, rappresenta un "segnaposto" (un attore) destinato a essere sostituito da un individuo del team. Il lavoratore è caratterizzato da particolari responsabilità e capacità come l'essere in grado di eseguire specifiche attività e sviluppare prestabiliti manufatti.
Consultare: Attività, Manufatto, Workflow.

Workflow (Flusso di lavoro) – *Workflow*

Realizzazione di una parte o di un intero caso d'uso di business. Può essere descritto in termini di diagramma delle attività, in cui si evidenziano i vari lavoratori, le attività che eseguono e i manufatti prodotti.
Consultare: Attività, Manufatto, Worker.

X

XP eXtreme programming

Processo leggero di sviluppo del software nel quale viene conferita particolare importanza al processo di implementazione (code-centric).