UML Tutorial - Italiano

Luiggi

0.1 UML Tutorial

Il linguaggio di modellazione unificato (UML) è diventato velocemente lo standard nella costruzione di software Object Oriented. Le specifiche dell'Object Management Group (OMG) affermano che:
"UML è un linguaggio grafico per la visualizzazione, specifica, costruzione e documentazione di artefatti di un sistema software. UML offre un metodo standard di scrivere una planimetria di sistema, che permette di specificare cose come processi aziendali, funzioni di sistema o addirittura cose più concrete come stati di linguaggi di programmazione, schemi di database e componenti software riutilizzabili."
La cosa importante è dunque che UML è un linguaggio per specificare e non un metodo o procedura. L'UML è usato per definire un sistema software, per descrivere gli artefatti nel sistema, per documentare e costruire - è il linguaggio in cui è scritta la planimetria. L'UML può essere usato in vari modi per supportare le metodologie dello sviluppo del software, ma di per se non specifica queste metodologie o processi. Semplicemente l'UML definisce le notazioni e la semantica per i seguenti domini:

- Interazione utente o Use Case Model - descrive i limiti e le interazioni tra sistema e utenti. In qualche modo corrisponde al modello dei requisiti.
- Interazione o Communication Model - descrive come gli oggetti nel sistema interagiranno tra loro per svolgere i loro compiti.
- Stato o Dynamic Model - diagrammi di stato che descrivono appunto gli stati o le condizioni che le varie classi assumono nel tempo. I diagrammi di attività descrivono il flusso del lavoro che il sistema implementerà.
- Logico o Class Model - descrive le classi e oggetti che comporranno il sistema.
- Il modello dei componenti fisici (Physical Component Model) - descrive il software (e a volte anche componenti hardware) che costituiscono il sistema.
- Il modello di distribuzione fisico (Physical Deployment Model) - descrive l'architettura fisica e la distribuzione delle componenti su una specifica architettura hardware.

0.2 UML 2.0

L'UML 2 si basa sullo standard dell'UML già di successo 1.x, che è diventato un'industria standard per la modellazione, design e costruzione di sistemi software così come processi aziendali e scientifici più generalizzati. UML 2 definisce 13 tipi base di diagrammi divisi in due insiemi generali:

0.2.1 Diagrammi di modellistica strutturale (Structural Modeling Diagrams)

Essi definiscono l'architettura statica di un modello. Sono utilizzati per modellare le 'cose' che costituiscono il modello quali classi, oggetti, interfacce e componenti fisiche. In più loro sono usati per modellare le relazioni e le dipendenze tra elementi.

- Diagrammi Package - sono usati per dividere il modello in contenitori logici o 'package' e descrivono le interazioni tra essi ad alti livelli.
- Classi o diagrammi strutturati - definiscono gli elementi di base di un modello: i tipi, le classi e i materiali generali che sono usati per costruire un modello completo.
- Diagrammi Oggetti - mostrano quante istanze di elementi strutturali sono relazionati e usati a tempo di esecuzione (run-time)
- Struttura composita - diagrammi che forniscono un mezzo per stratificare la struttura di un elemento e concentrarsi su dettagli interni, costruzione e relazioni.
- Diagrammi delle componenti - sono usati per modellare strutture più complesse o a livelli alti, spesso sono costituite da una o più classi e costituiscono un'interfaccia ben definita.
- Diagrammi di distribuzione - mostrano le disposizioni fisiche di artefatti significativi all'interno di un ambiente del mondo reale.

0.2.2 Diagrammi di modellistica comportamentale (Behavioral Modeling Diagrams)

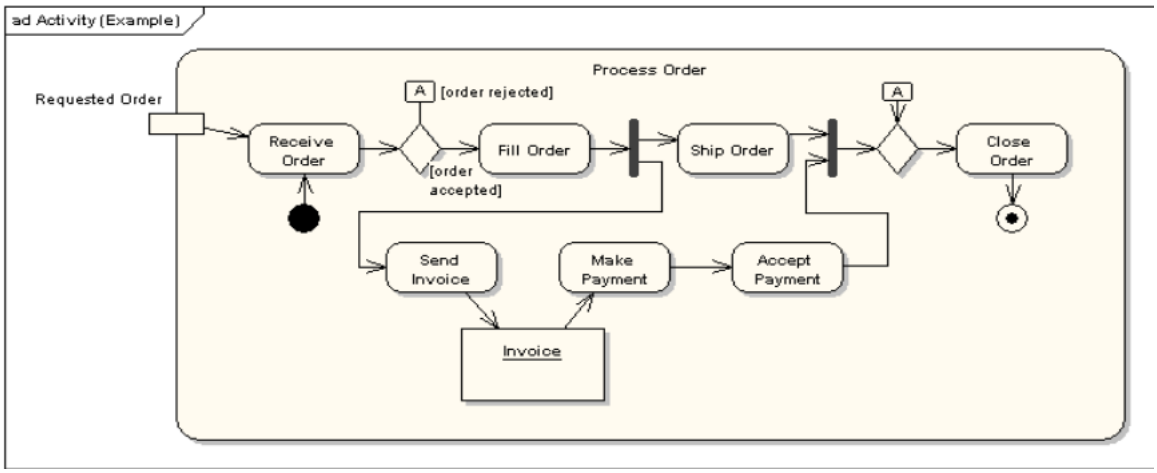
I diagrammi comportamentali catturano le varietà di interazioni e stati simultanei all'interno di un modello eseguito 'nel tempo'.

- Diagrammi dei casi d'uso - sono usati per modellare le interazioni utenti/sistema. Definiscono il comportamento, i requisiti e i vincoli nella forma di copioni e scenari.
- Diagrammi delle attività - hanno un ampio numero di utilizzi, dalla definizione basica del flusso del programma, alla cattura dei punti decisionali e azioni all'interno di un qualsiasi processo generalizzato.
- Diagrammi delle macchine a stati - sono essenziali per la comprensione, istante per istante, della condizione o dello stato di esecuzione di un modello mentre esso è eseguito.
- Diagrammi di comunicazione - mostrano il network e le sequenze di messaggi o comunicazioni tra oggetti a tempo di esecuzione (run-time) durante un'istanza di collaborazione.
- Diagrammi di sequenza - sono molto correlati con i diagrammi di comunicazione e mostrano la sequenza di messaggi passati tra oggetti usando una linea di tempo verticale.
- Diagrammi di tempo - fondono i diagrammi di sequenza e i diagrammi di Stato per fornire una vista dello stato di un oggetto nel tempo e i messaggi che ne modificano lo stato stesso.
- Diagrammi di panoramica delle interazioni - fondono i diagrammi di attività e di sequenza per far sì che i frammenti di interazione possano essere facilmente combinati con punti e flussi decisionali.

0.3 Diagramma delle attività

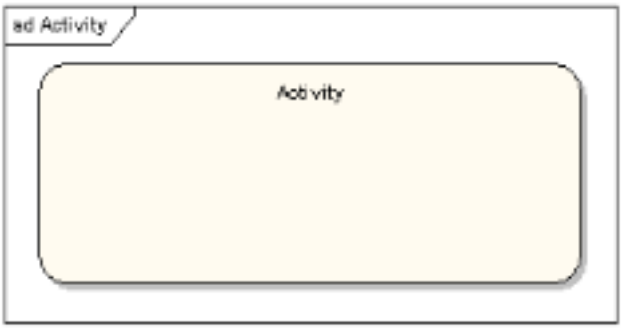
0.3.1 Diagrammi di attività

Nell'UML i diagrammi di attività sono usati per mostrare la sequenza delle attività. Questo diagramma mostra il flusso di lavoro dall'inizio alla fine dettagliando i vari percorsi decisionali che esistono in una progressione di eventi contenuta nell'attività. Essi possono essere usati per dettagliare situazioni dove potrebbero avvenire processi paralleli durante l'esecuzione di diverse attività. Il diagramma di attività è utile per la modellazione aziendale. Un esempio:



0.3.2 Attività

Un'attività è la specifica di una sequenza comportamentale parametrizzata. Un'attività è mostrata come un rettangolo stondato che racchiude tutte le azioni, flusso di controllo e altri elementi che costituiscono l'attività.

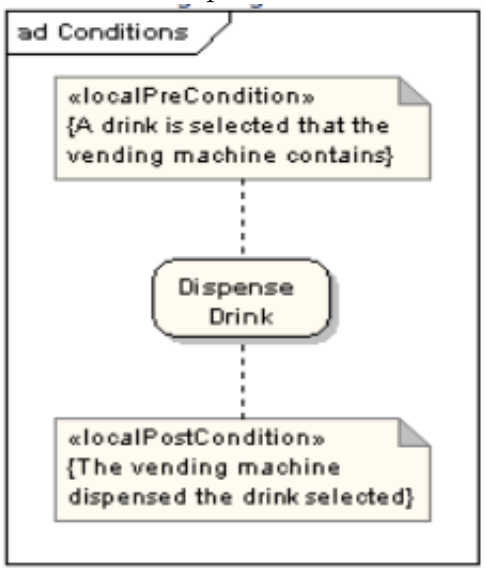


0.3.3 Azioni

Un'azione rappresenta un singolo pasto dentro un'attività. Le azioni sono denotate da rettangoli stondati.

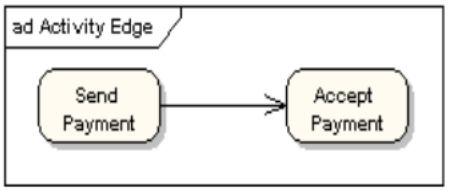
0.3.4 Vincoli di azione

I vincoli possono essere attaccati all'azione. Il seguente diagramma mostra un'azione con precondizioni e postcondizioni locali.



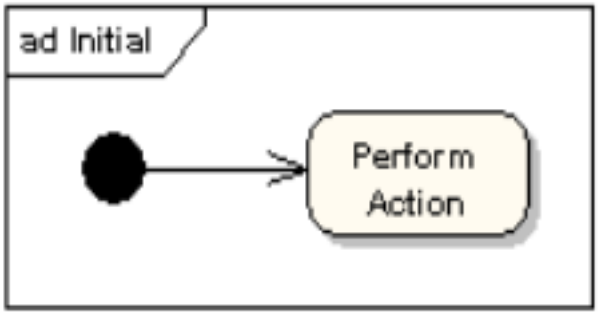
0.3.5 Flusso di controllo

Il flusso di controllo mostra il flusso appunto di un'azione prima di procedere alla successiva. La sua notazione è costituita da una freccia.



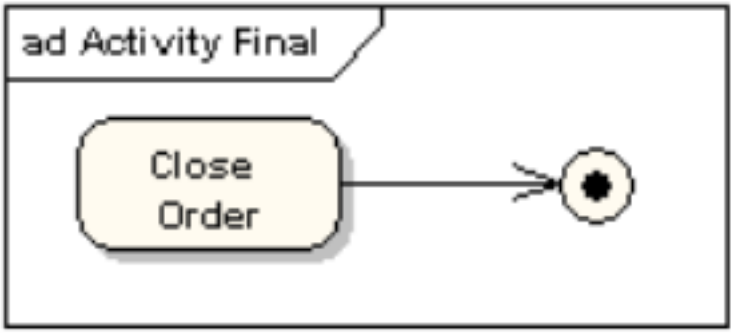
0.3.6 Nodo iniziale

Il nodo iniziale è fatto da un pallino nero come qui sotto:

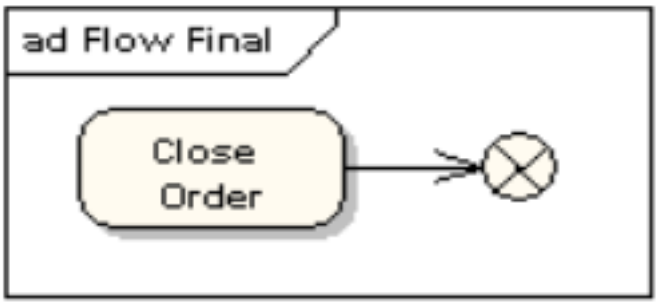


0.3.7 Nodo finale

Ci sono sostanzialmente due tipi di nodi finali: Il nodo di fine attività e di fine flusso. Il nodo di fine attività è formato da un pallino nero dentro un cerchio:



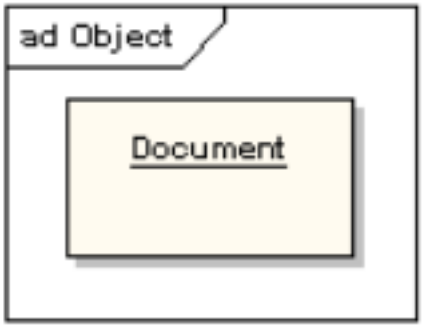
Mentre il nodo di fine flusso è fatto con un cerchio con una x in mezzo:



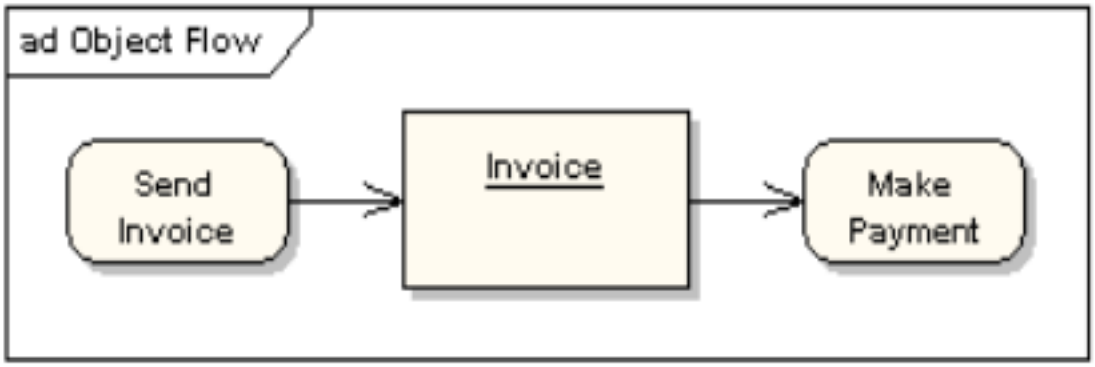
La differenza tra i due è sostanzialmente che il nodo di fine flusso denota la fine di un singolo flusso di controllo, mentre il nodo di fine attività determina la fine di tutti i flussi di controllo all'interno dell'attività.

0.3.8 Oggetti e flussi oggetto

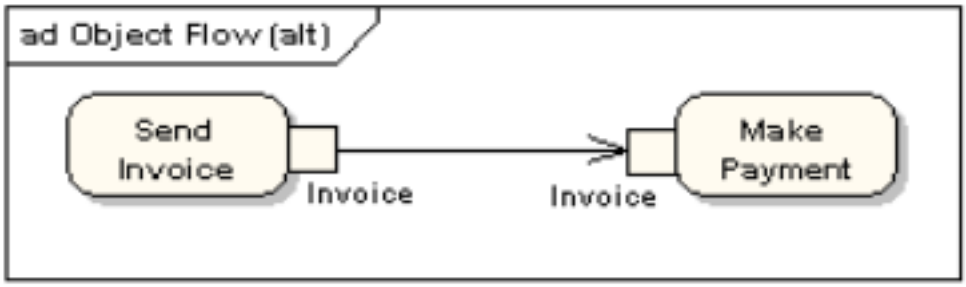
Un flusso oggetto è un percorso lungo i quali gli oggetti o dati possono passare. Un oggetto è denotato da un rettangolo:



Un flusso oggetto è un connettore con una freccia che denota la direzione dell'oggetto passato:



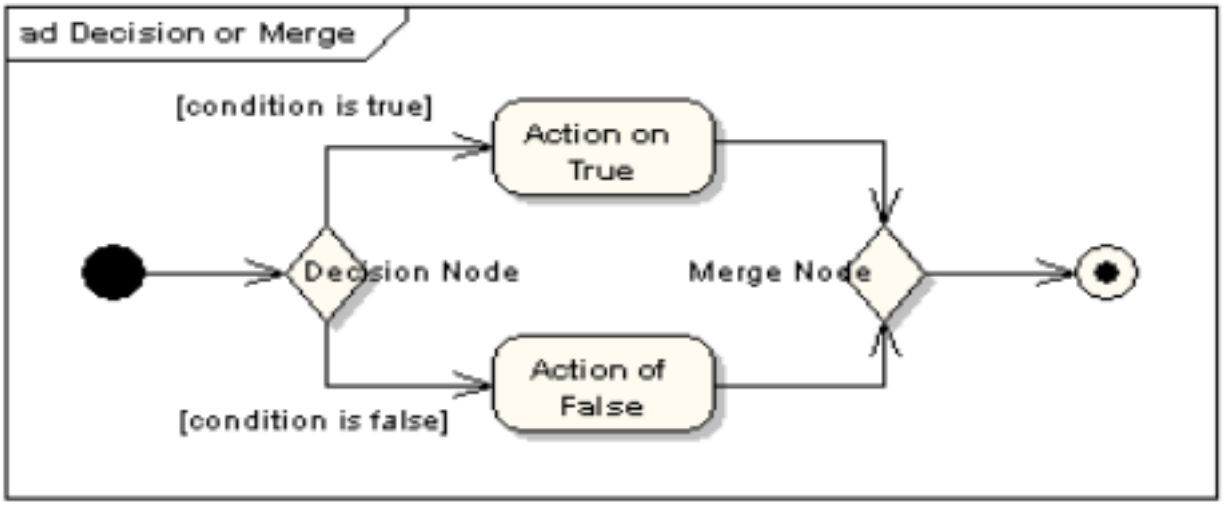
Un flusso oggetto deve avere un oggetto in almeno una delle estremità. Una breve notazone per il diagramma di sopra che userebbe dei pin di input e output:



Un magazzino di dati è disegnato come un oggetto con il keyword «datastore»:

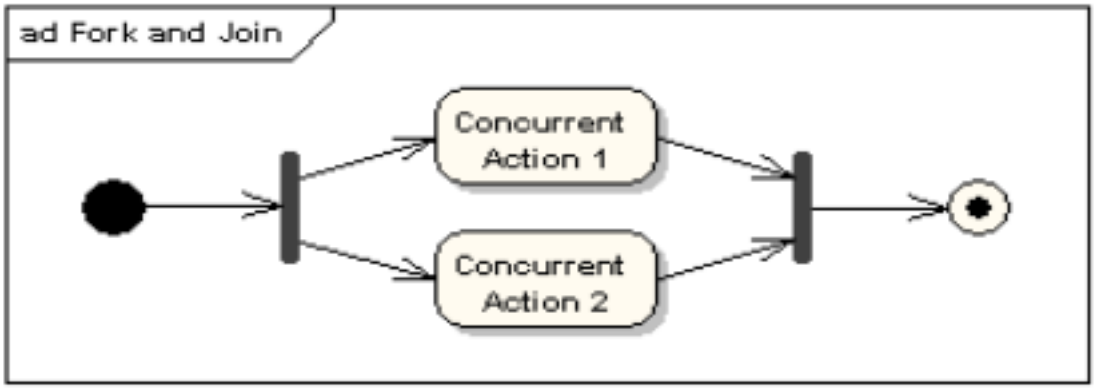
0.3.9 Nodi di decisione e fusione

I nodi di decisione e fusione hanno la stessa denotazione: un rombo. Entrambi possono avere un nome. Un flusso di controllo che esce da un nodo di decisione avrà una guardia condizionale che permetterà al controllo di direzionare il flusso se la guardia è rispettata:



0.3.10 Nodi Fork e Join

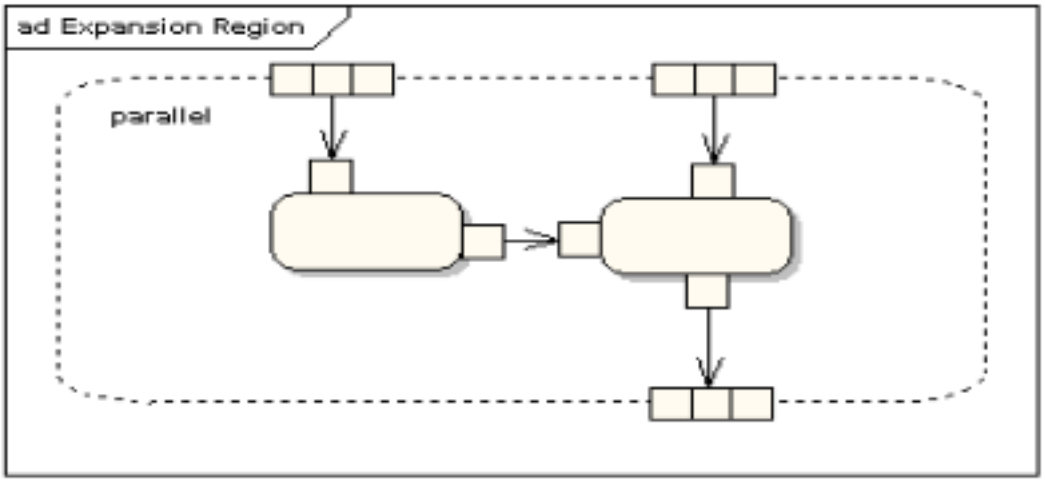
Le Fork e le Join hanno la stessa notazione: una barra verticale o orizzontale nera (a seconda dell'orientamento top-down o left-right). Esse indicano l'inizio e la fine di controlli concorrenti:



Una join è diversa da una merge nel fatto che la join sincronizza 2 flussi entranti e produce un singolo flusso uscente, mentre una merge seleziona uno dei due flussi entranti a seconda della condizione.

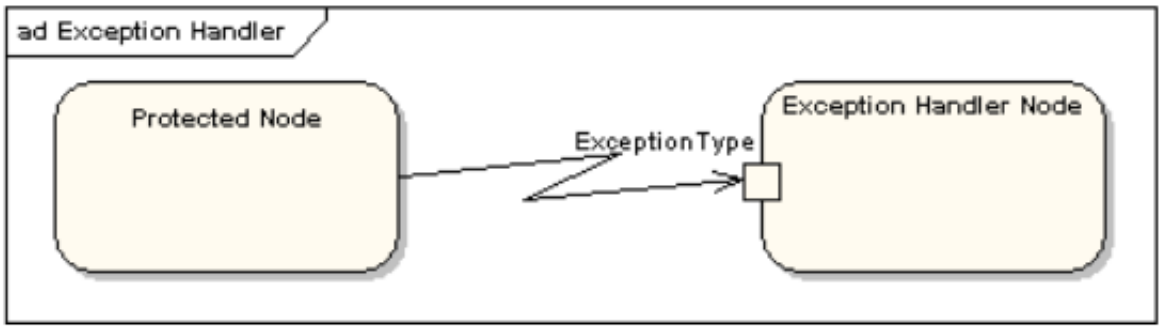
0.3.11 Regione di espansione

Una regione di espansione è una regione strutturata di attività che viene eseguita più volte. I nodi di input e output di espansione sono disegnati come un gruppo di tre box che rappresentano una selezione multipla di elementi. La parola chiave iterative, parallel o stream è mostrata nell'angolo superiore della regione.



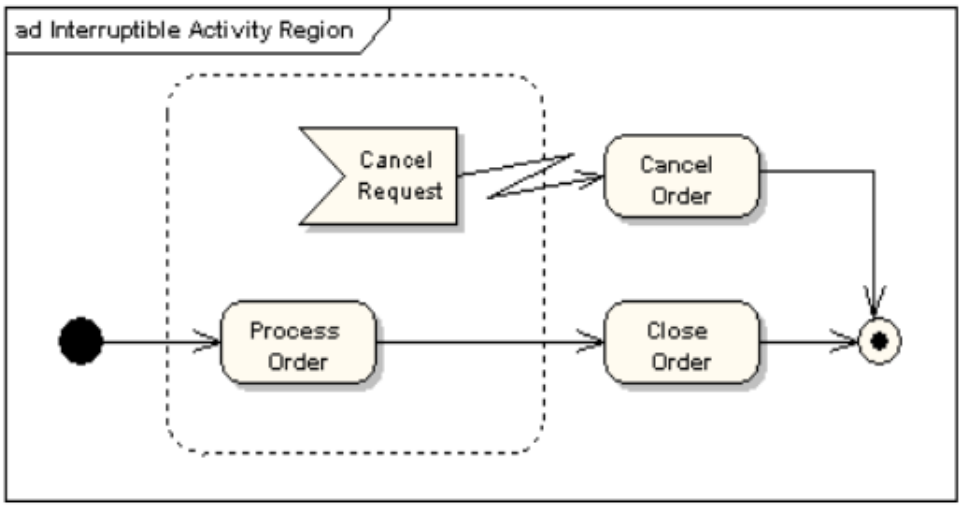
0.3.12 Gestore di eccezioni

Un gestore di eccezioni può essere modellato nel diagramma di attività come segue:



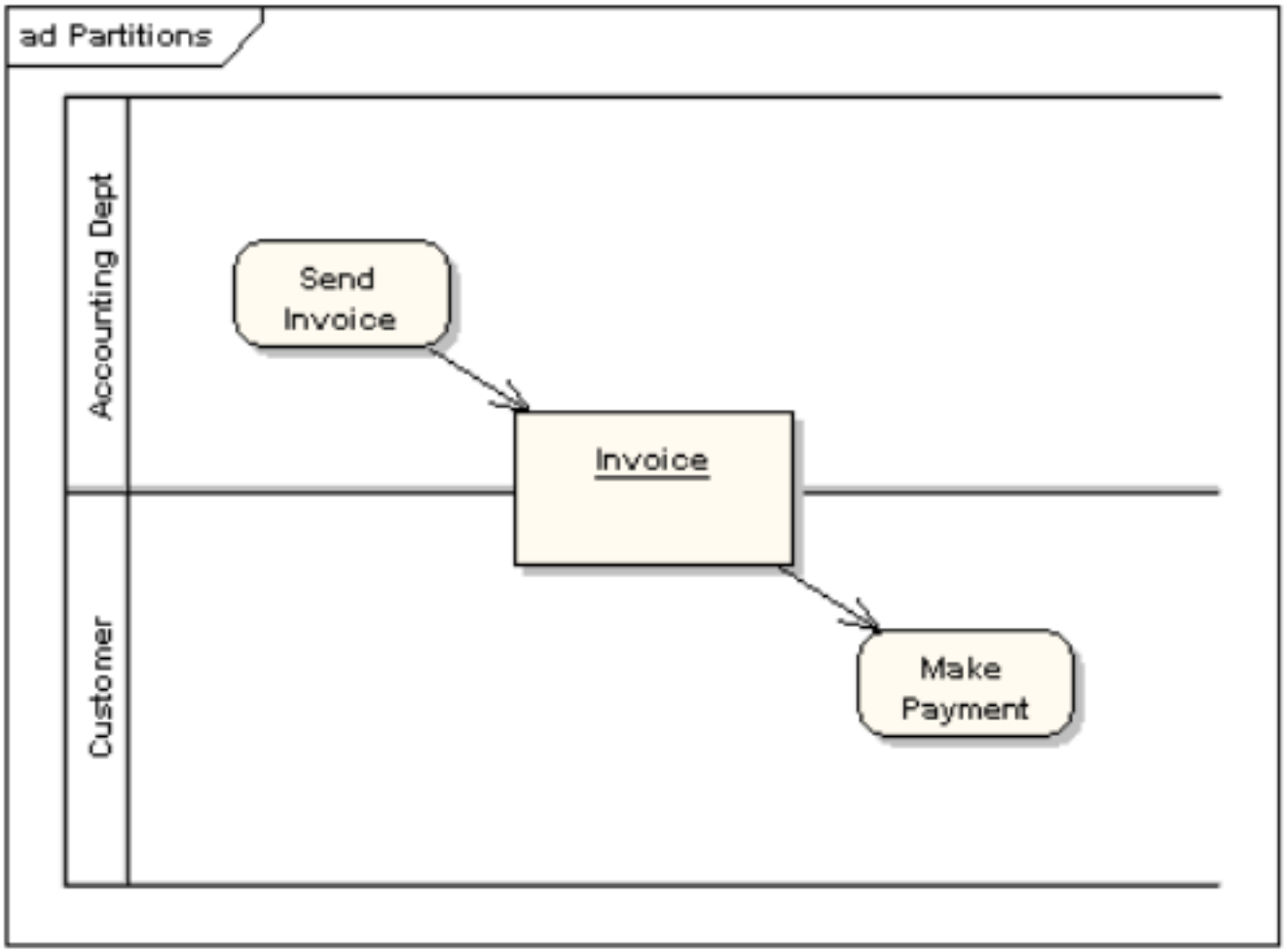
0.3.13 Regione di attività interrompibile

Una regione di attività interrompibile circonda un gruppo di azioni che possono essere interrotte. Nel semplice esempio sotto, l'azione Order sarà eseguita fino a che non sarà completata mentre passa all'azione Close Order, finchè un'azione Cancel Request è ricevuta, che passerà il controllo all'azione Cancel Order.



0.3.14 Partizione

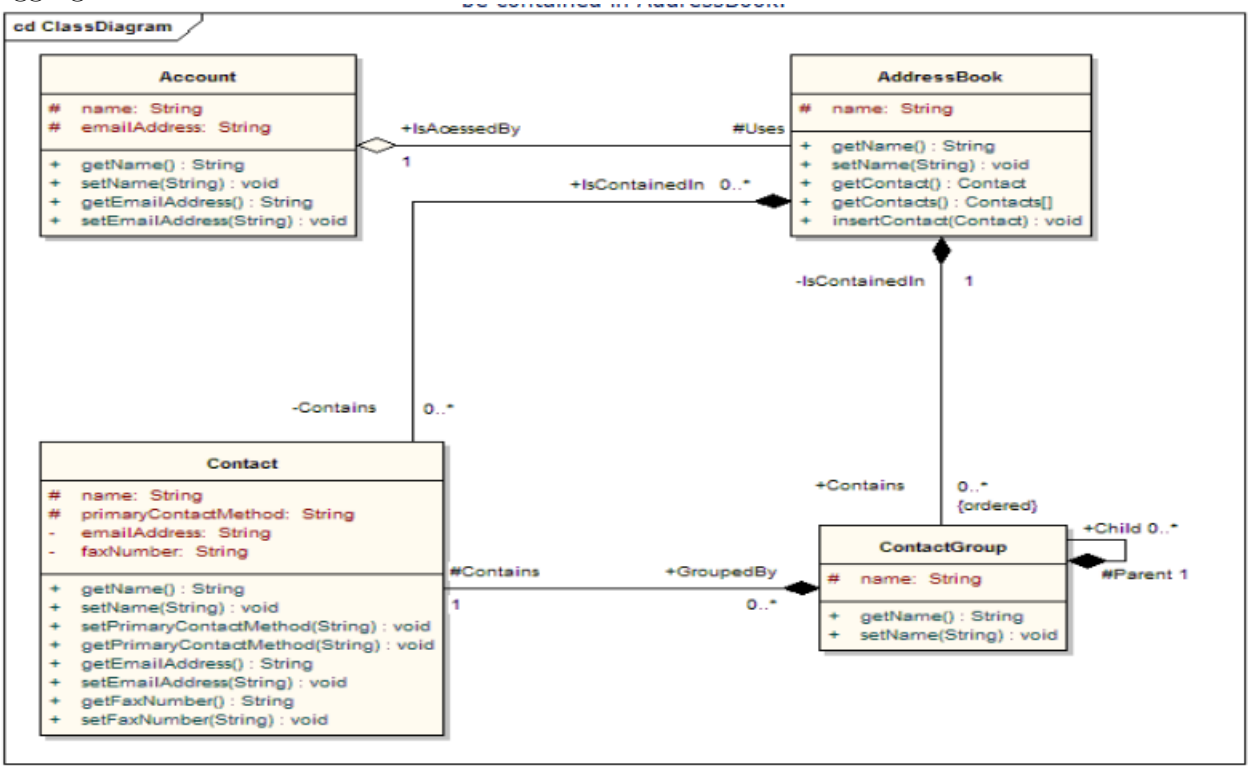
L'attività partizione è mostrata sia orizzontalmente che verticalmente. Nel diagramma seguente, le partizioni sono utilizzate per separare le azioni all'interno di un'attività in quelle eseguite dal reparto contabilità e quelle eseguite dal cliente.



0.4 Diagramma delle classi

0.4.1 Diagrammi delle classi

Il diagramma delle classi mostra i blocchi che compongono qualsiasi sistema orientato agli oggetti. Il diagramma delle classi descrive la vista statica del modello o parte del modello, specificando quali attributi e comportamenti ha, mostrando i metodi per svolgere le operazioni. I diagrammi delle classi sono i più utili per illustrare le relazioni tra classi e interfacce, composizioni o utilizzi, connessioni. Il diagramma sotto illustra relazioni di aggregazione tra classi.

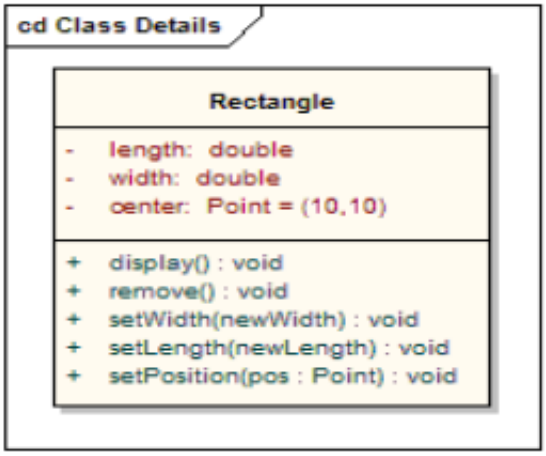


0.4.2 Classi

Una classe è un elemento che definisce gli attributi e comportamenti che un oggetto è capace di generare. Il comportamento è quello descritto dai possibili messaggi che la classe è in grado di capire attraverso le operazioni che sono appropriate per ogni messaggio. Le classi possono anche contenere le definizioni di vincoli con valori etichettati e stereotipi.

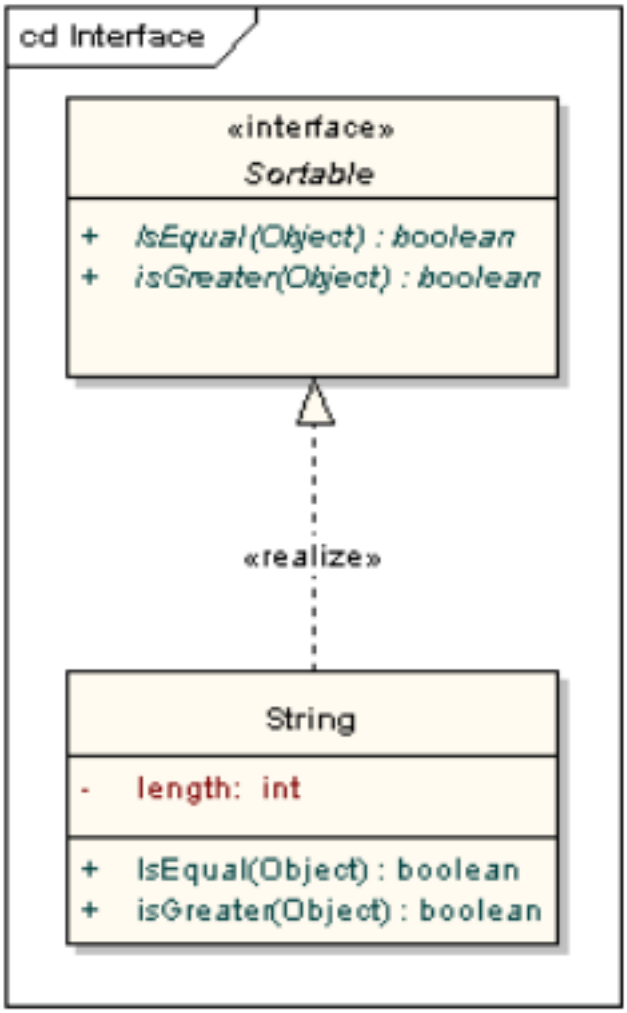
0.4.3 Notazione di classe

Le classi sono rappresentate da rettangoli che mostrano il nome della classe e opzionalmente il nome delle operazioni e attributi. I compartimenti sono usati per suddividere nome della classe, attributi e operazioni. In più potrebbero esserci vincoli, valori iniziali e parametri che potrebbero essere assegnati alle classi.



0.4.4 Interfacce

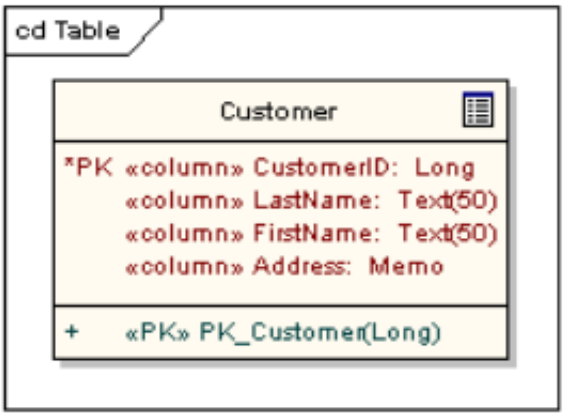
Un'interfaccia è una specifica di comportamento da implementare, è un contratto. Realizzando un'interfaccia, le classi garantiscono di supportare un comportamento richiesto, che permette al sistema di trattare elementi non relazionati allo stesso modo (attraverso la stessa interfaccia).



Le interfacce possono essere disegnate in maniera simile alle classi, con anche le operazioni, come mostrato sopra. Esse possono essere anche disegnate come cerchi con operazioni esplicite dettagliate. Quando sono circolari, non hanno la freccia di Realize.

0.4.5 Tabelle

Una tabella è uno stereotipo di classe. È disegnata con una piccola icona di tabella nell'angolo destro in alto. Gli attributi hanno gli stereotipi «column» e seguono le specifiche dei database (chiavi primarie PK, FK ...).

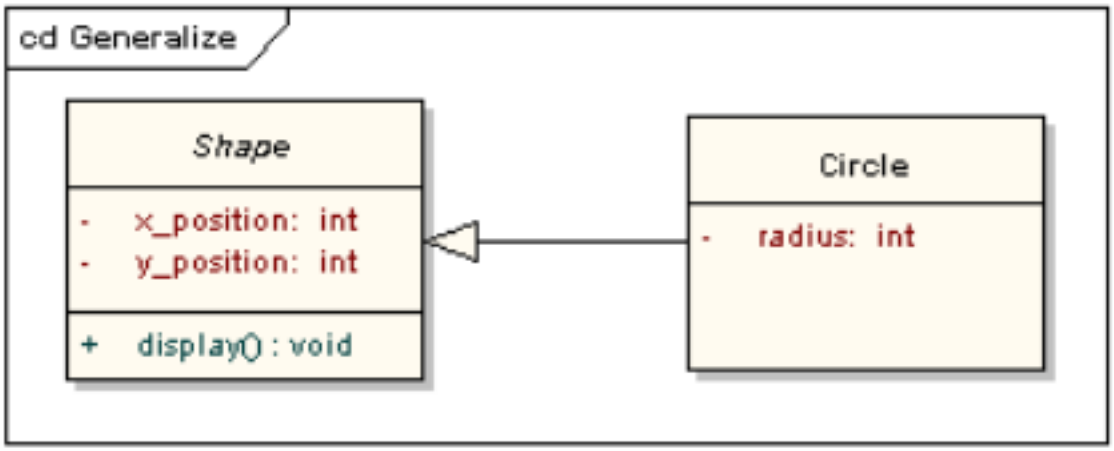


0.4.6 Associazioni

Un'associazione tra due elementi implica una relazione, spesso implementata come una variabile in una delle classi. Questo connettore potrebbe includere il nome del ruolo al centro o nelle estremità, con anche la cardinalità, la direzione e i vincoli. L'associazione è la relazione generale tra elementi.

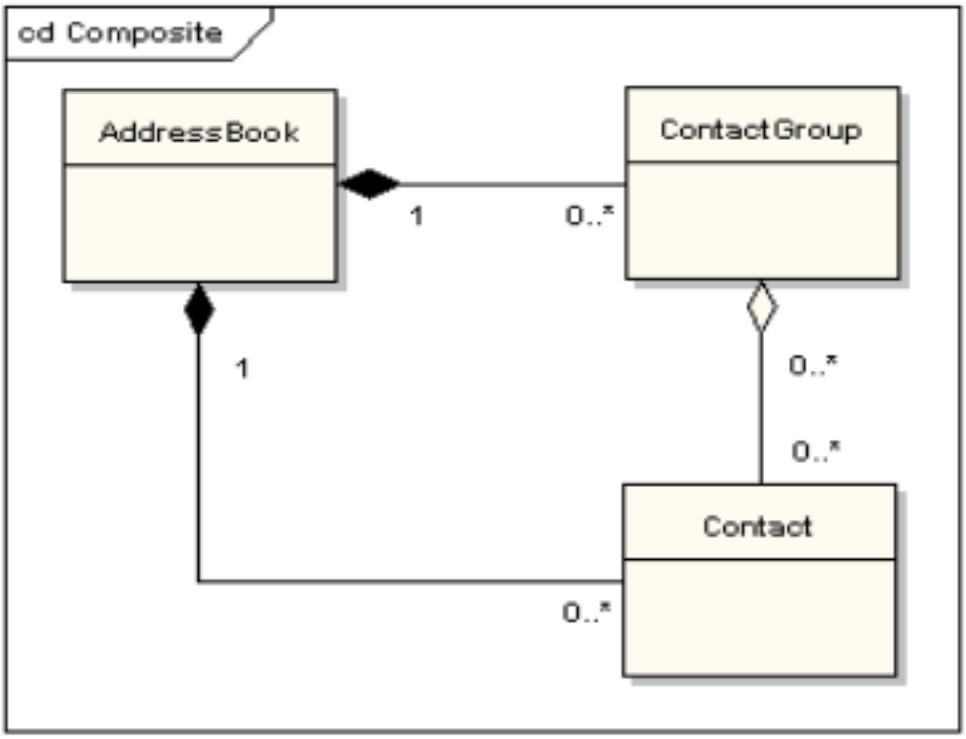
0.4.7 Generalizzazioni

Una generalizzazione è usata per indicare l'ereditarietà tra classi. Disegnata da un classificatore specifico a un classificatore generalizzato, indica appunto una sottoclasse derivata da una classe "padre". Il figlio eredita tutto dal padre.



0.4.8 Aggregazioni

Le aggregazioni sono usate per descrivere elementi che sono fatti da componenti più piccoli. La relazione di aggregazione è debole, ed è costituita da un rombo bianco che punta alla classe padre. Una forma più forte di aggregazione, ovvero la composizione, è rappresentata da un rombo nero. Se la classe padre della composizione è cancellata, allora anche tutte le sue "eredità" lo sono (non può esistere la parte senza il tutto).



0.4.9 Dipendenze

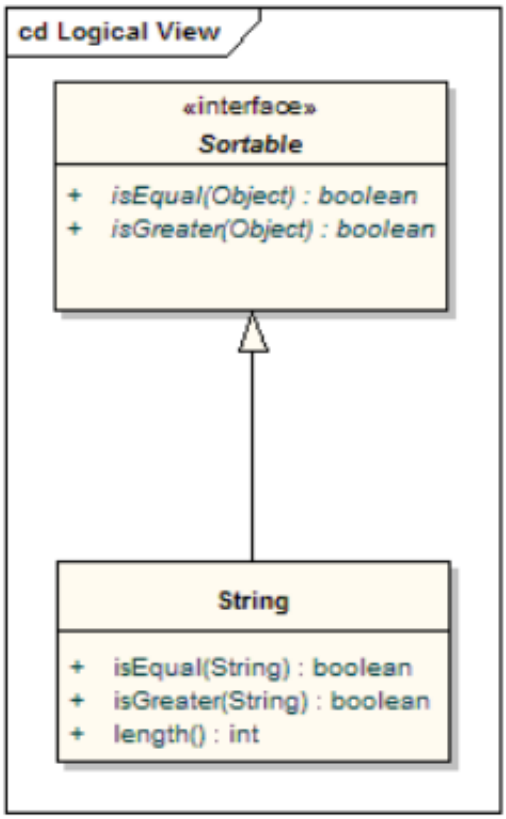
Una dipendenza è usata per modellare un ampio range di relazioni dipendenti tra modelli di elementi. Sarebbe normalmente usata all'inizio del processo di Design, quando si sa che c'è un certo collegamento tra elementi ma è troppo presto per sapere esattamente che tipo di relazione sia. Più avanti nel processo, le dipendenze verranno poi stereotipate con «trace», «import», «instanciate» e altri.

0.4.10 Tracce

La relazione di traccia è una specializzazione di dipendenza, che collega elementi del modello o insiemi di elementi che rappresentano la stessa idea attraverso i modelli. Le tracce sono spesso usate per tracciare i requisiti e modellare i cambiamenti. Quando occorre un cambiamento in entrambe le direzioni, l'ordine di dipendenze è spesso ignorato. Le proprietà della relazione possono specificare la mappatura della traccia, ma la traccia è generalmente bidirezionale, informale e raramente calcolabile.

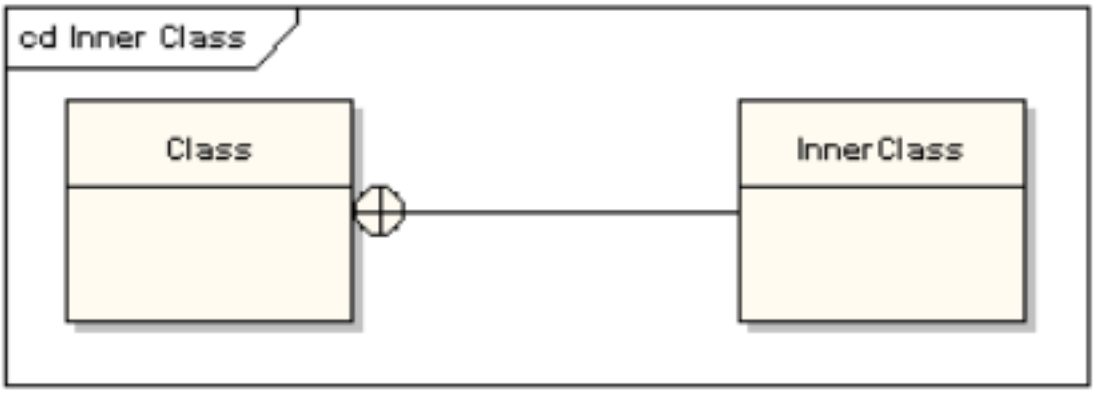
0.4.11 Realizzazioni

L'oggetto sorgente implementa o realizza la destinazione. Realize viene utilizzato per esprimere la tracciabilità e la completezza del modello: un processo aziendale o un requisito viene realizzato da uno o più casi d'uso che a loro volta vengono realizzati da alcune classi, che a loro volta vengono realizzati da un componente, ecc. La mappatura dei requisiti, classi, ecc. attraverso la progettazione del sistema, fino ai livelli di astrazione della modellazione, assicura che l'immagine generale del sistema si ricordi e rifletta tutte le piccole immagini e i dettagli che la vincolano e la definiscono. Una realizzazione viene mostrata come una linea tratteggiata con una punta di freccia solida e lo stereotipo «realize».



0.4.12 Annidamenti

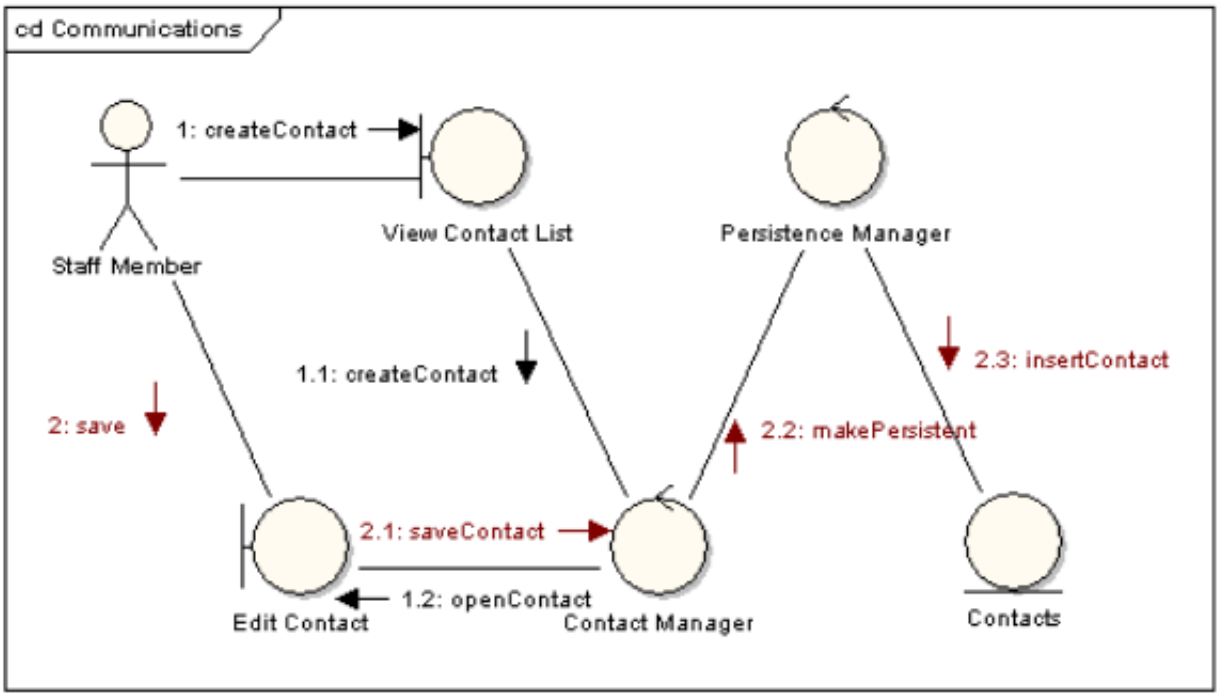
Un annidamento è un connettore che mostra che l'elemento sorgente è annidato dentro l'elemento target. È in pratica la definizione schematica del concetto di Inner Class di Java.

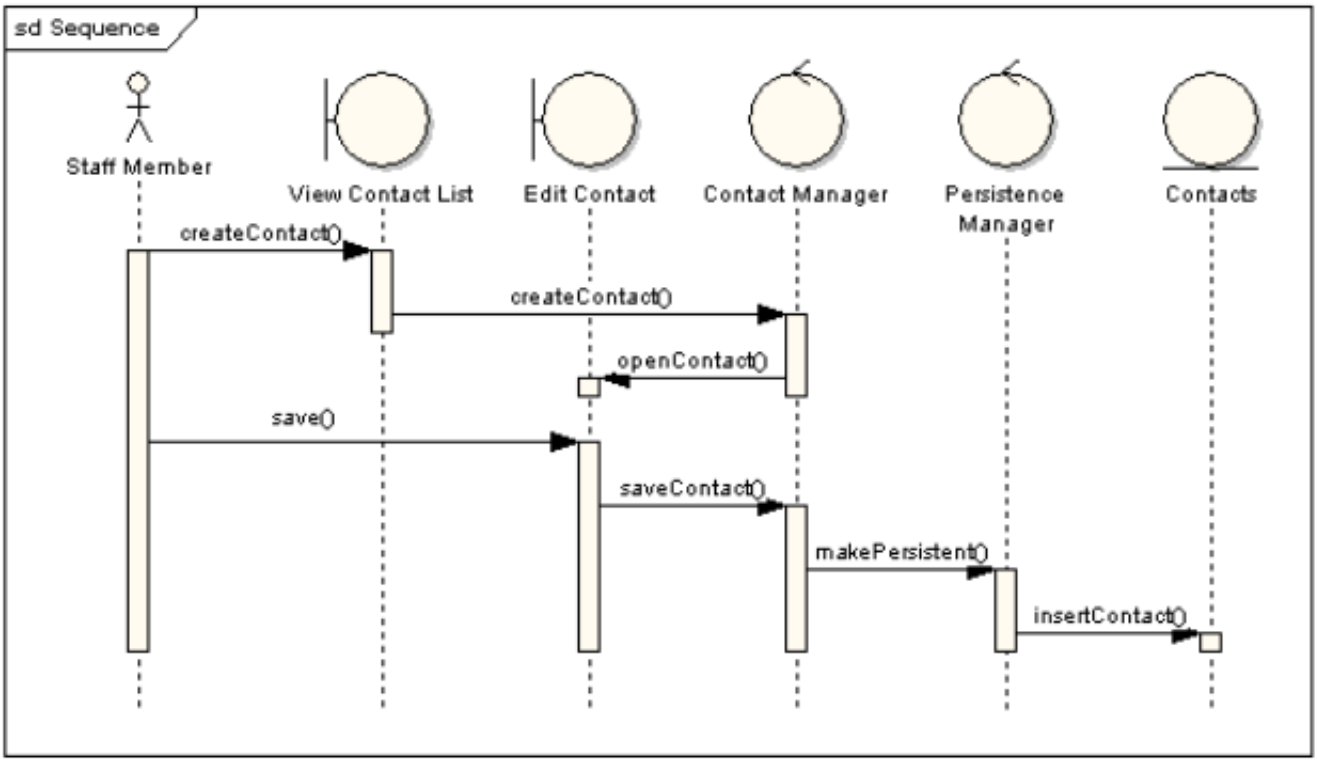


0.5 Diagramma di comunicazione

0.5.1 Diagrammi di comunicazione

Un diagramma di comunicazione, formalmente chiamato diagramma di collaborazione, è un diagramma di interazioni che mostra informazioni simili a un diagramma di sequenza ma è concentrato sulle relazioni tra oggetti. Sui diagrammi di comunicazione, gli oggetti sono mostrati con connettori di associazione. I messaggi sono aggiunti alle associazioni e mostrano delle freccette che puntano in direzione del flusso del messaggio. La sequenza di messaggi è numerata per oggetti adiacenti.

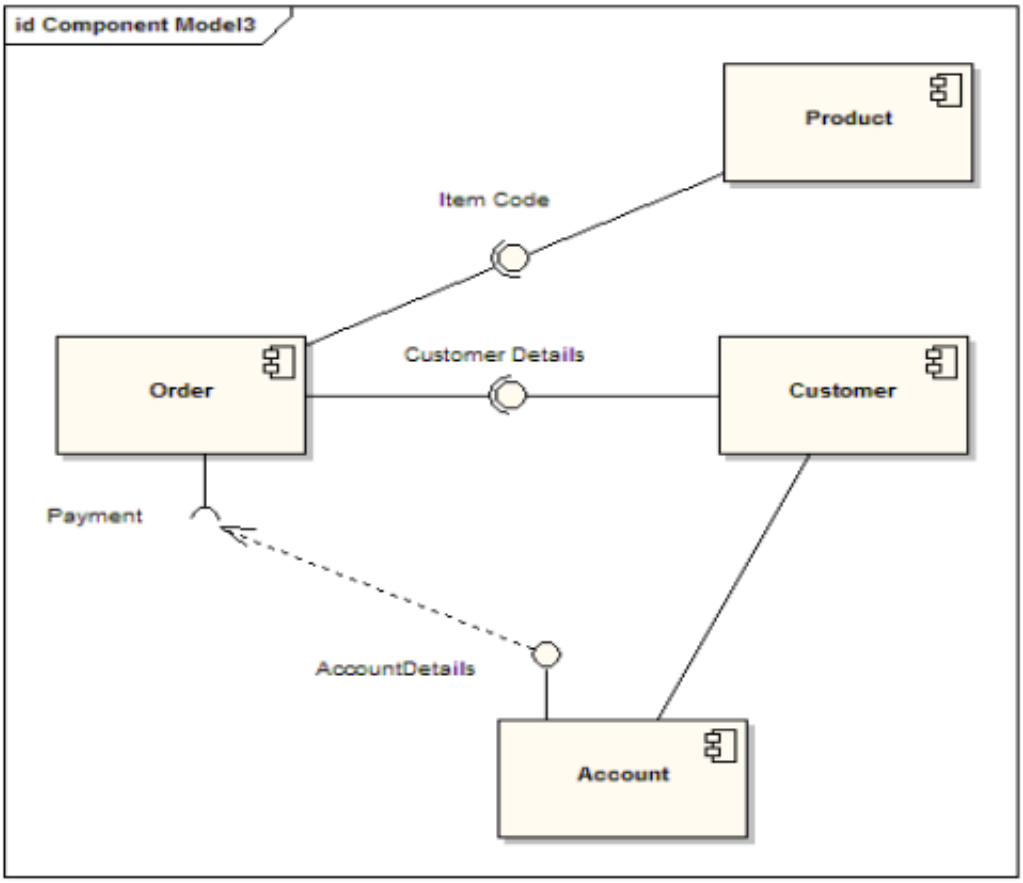




0.6 Diagramma delle componenti

0.6.1 Diagrammi delle componenti

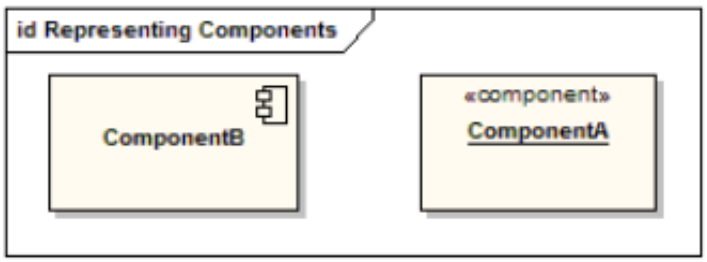
I diagrammi delle componenti mostrano i pezzi di software, controlli incorporati,etc. che costituiranno il sistema. Un diagramma delle componenti ha un più alto livello di astrazione rispetto a quello delle classi - spesso un componente è implementato da una o più classi (o oggetti) a tempo di esecuzione. Sono elementi costitutivi, tant'è che alla fine un componente può comprendere gran parte di un sistema. Nell'esempio sotto, i connettori di assemblaggio 'collegano' le interfacce fornite dal Prodotto e dal Cliente alle interfacce richieste specificate dall'Ordine. Una relazione di dipendenza associa i dettagli dell'account associato di un cliente all'interfaccia richiesta, "Pagamento", indicata dall'ordine.



I componenti sono simili nella pratica ai diagrammi dei package, nella definizione dei limiti e sono usati per gruppi di elementi in strutture logiche. La differenza tra i due diagrammi sostanzialmente sta nel fatto che il diagramma delle componenti offre un meccanismo di raggruppamento semanticamente più ricco. Inoltre, nel diagramma delle componenti tutti gli elementi del modello sono privati mentre i diagrammi dei package mostrano solo elementi pubblici.

0.6.2 Come si rappresentano le componenti

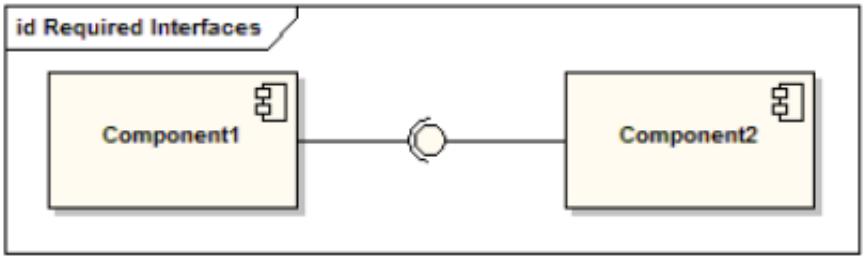
Le componenti si rappresentano con classificatori rettangolari con la parola chiave «component», e opzionalmente potrebbe essere disegnato con un rettangolo con l'icona di componente in alto a destra.



0.6.3 Interfacce richieste

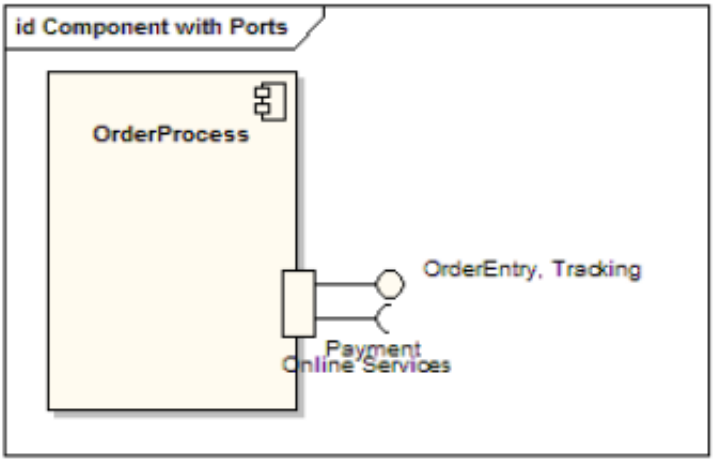
Il connettore di assemblamento collega l'interfaccia richiesta dalla componente con l'interfaccia fornita da un'altra componente; questo permette alla componente di fornire i servizi che

un'altra componente richiede. Le interfacce sono collezioni di uno o più metodi che possono opzionalmente avere anche degli attributi.



0.6.4 Componenti con porti

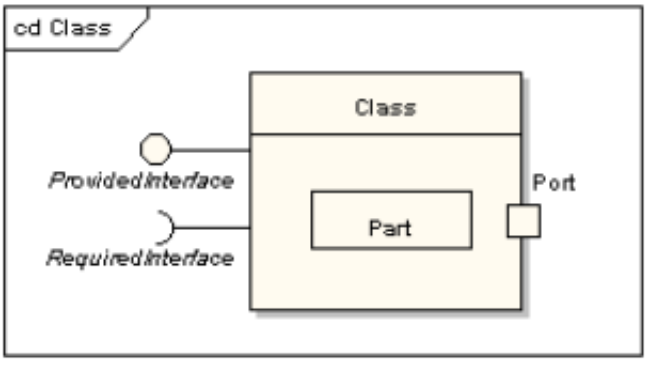
Usare i porti nei diagrammi delle componenti permette a un servizio o comportamento di essere specificato nel suo ambiente come un servizio o comportamento che un componente richiede. I porti potrebbero specificare input/output come se operassero bidirezionalmente:



0.7 Diagramma di struttura composita

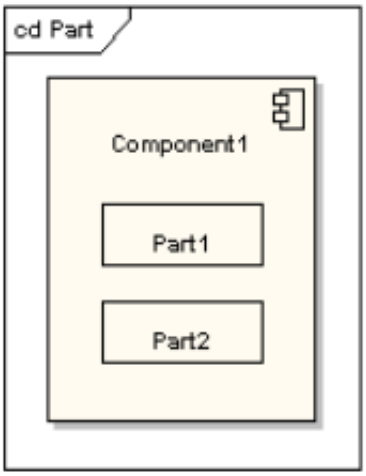
0.7.1 Diagrammi compositi

Un diagramma di struttura composita è un diagramma che mostra la struttura interna di un classificatore, includendo i suoi punti di interazione con altre parti del sistema. Mostra la configurazione e la relazione delle parti che insieme svolgono il compito del classificatore contenuto.



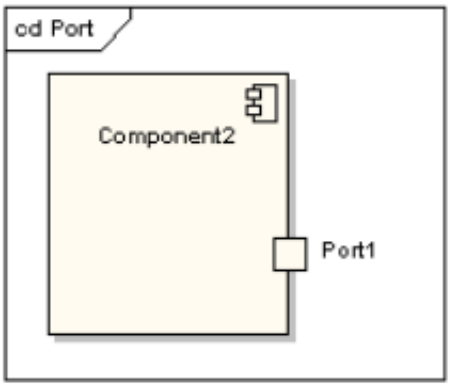
0.7.2 Parte

Una parte è un elemento che rappresenta un insieme di una o più istanze che appartengono a un'istanza di classificatore interna. Quindi ad esempio, se l'istanza di un diagramma aveva un insieme di elementi grafici, allora gli elementi grafici possono essere rappresentati come parti, se fosse utile farlo per modellare un qualche tipo di relazione tra loro. Si noti che una parte può essere rimossa dal suo genitore prima che il genitore venga eliminato, in modo che la parte non venga eliminata contemporaneamente. Una parte è disefnata come un rettangolo semplice contenuto dentro il corpo della classe o componente.



0.7.3 Porto

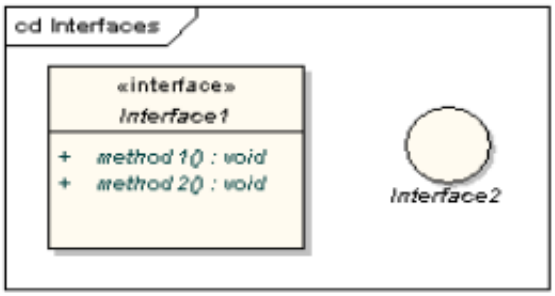
Un porto è un elemento tipizzato che rappresenta una parte visibile esternamente di un'istanza del classificatore che contiene elementi. I porti definiscono l'interazione tra un classificatore e il suo ambiente. Un port può apparire sul confine di una parte contenuta, una classe o una struttura composita. Un porto può specificare i servizi forniti da un classificatore nonché i servizi richiesti dal proprio ambiente. Esso è disegnato come un piccolo quadrato con un nome posto sul limite di un classificatore che lo possiede.



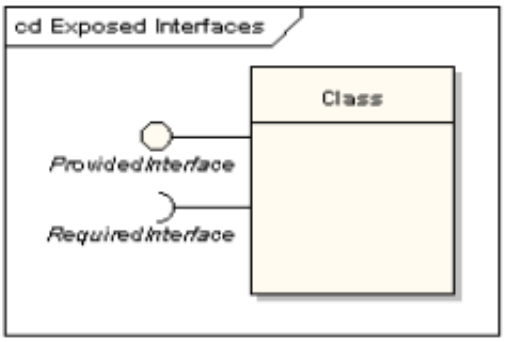
0.7.4 Interfacce

Un'interfaccia è simile a una classe ma con un po' di restrizioni. Tutte le operazioni delle interfacce sono pubbliche e astratte, e non forniscono alcun tipo di implementazione di default. Tutti gli attributi di un'interfaccia devono essere costanti. Tuttavia, mentre una classe può ereditare solo da una singola super classe, essa può implementare più

interfacce. Un'interfaccia può essere disegnata o come una classe con la parola chiave «interface» oppure con un cerchio.

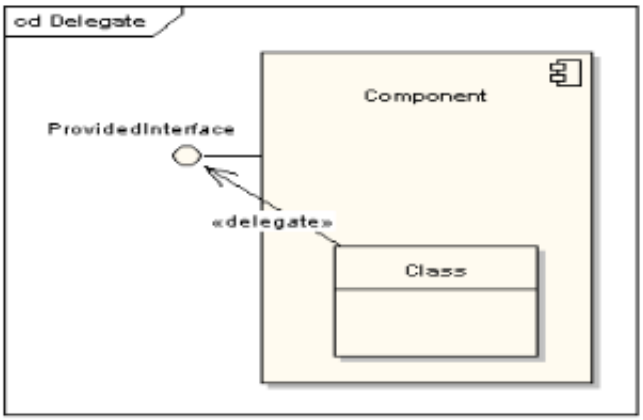


Si noti che la notazione cerchio non mostra le operazioni di interfaccia. Quando le interfacce vengono mostrate come appartenenti alle classi, vengono chiamate interfacce esposte. Un'interfaccia esposta può essere definita come fornita o richiesta. Un'interfaccia fornita è un'affermazione che il classificatore contenente fornisce le operazioni definite dall'elemento dell'interfaccia denominato ed è definito disegnando un collegamento di realizzazione tra la classe e l'interfaccia. Un'interfaccia richiesta è un'affermazione che il classificatore è in grado di comunicare con qualche altro classificatore che fornisce operazioni definite dall'elemento dell'interfaccia denominato ed è definito disegnando un collegamento di dipendenza tra la classe e l'interfaccia. Un'interfaccia fornita è mostrata come una palla su uno stecco, attaccata all'estremo dell'elemento classificatore. Un'interfaccia richiesta invece è mostrata come un semicerchio su uno stecco, sempre attaccata all'estremità del classificatore.



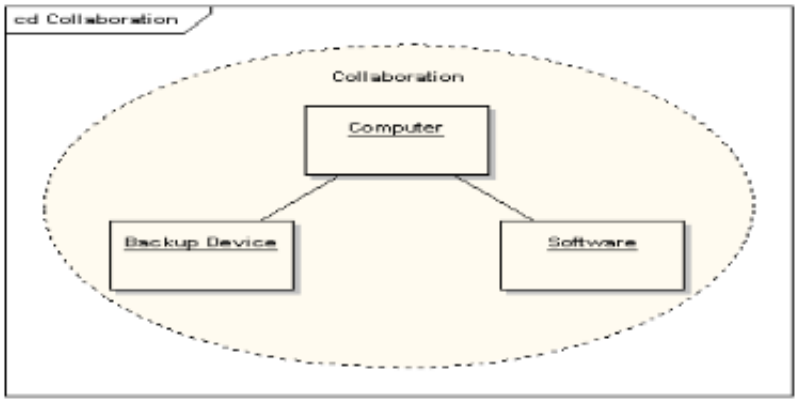
0.7.5 Delegato

Un connettore delegato è usato per definire i lavori interni dei porti e interfacce esterne di una componente. Esso è disegnato come una freccia con lo stereotipo «delegate». Collega un contratto esterno di un componente come mostrato dai suoi porti alla realizzazione interna del comportamento della parte del componente.



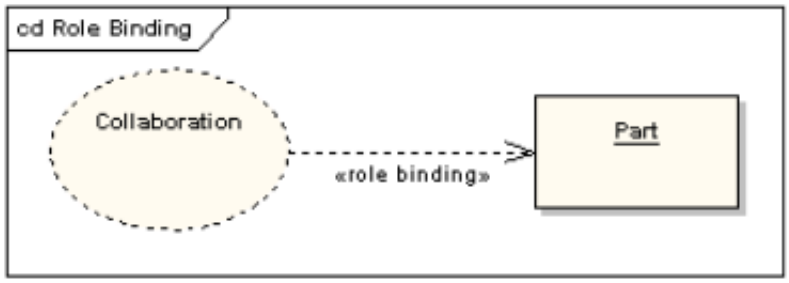
0.7.6 Collaborazione

Una collaborazione definisce un insieme di ruoli cooperativi usati collettivamente per illustrare una funzionalità specifica. Una collaborazione dovrebbe mostrare solo ruoli e attributi richiesti per portare a termine un lavoro o funzione definiti. L'isolamento dei ruoli primari è un esercizio per semplificare la struttura e chiarire il comportamento e prevede anche il riutilizzo. Una collaborazione spesso implementa uno schema. Una collaborazione è disegnato con un ellisse.



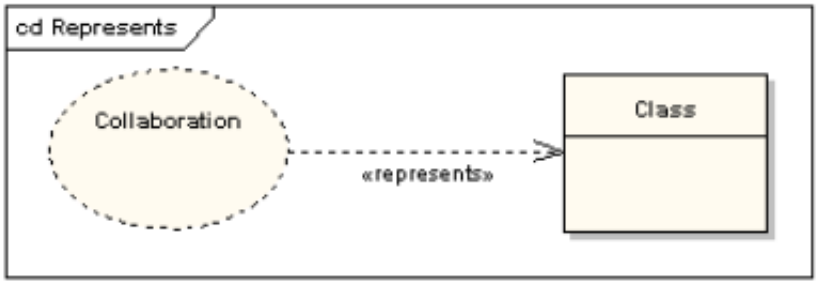
0.7.7 Associazione ruolo

Un collegamento di associazione ruolo è disegnato dalla collaborazione al classificatore che adempie al ruolo. È mostrato come una linea tratteggiata con una freccia e lo stereotipo «role binding».



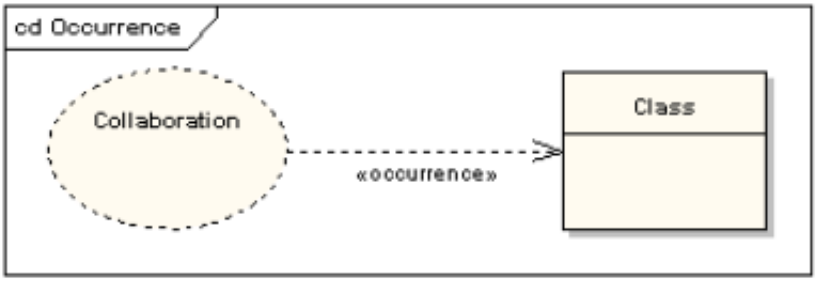
0.7.8 Represents

Un connettore 'represents' può essere disegnato da una collaborazione a un classificatore per mostrare che la collaborazione è usata in quest'ultimo. È disegnato come una linea tratteggiata, una freccia e lo stereotipo «represents».



0.7.9 Occorrenza

Un connettore occorrenza può essere disegnato da una collaborazione a un classificatore per mostrare che una collaborazione rappresenta il classificatore. È mostrata con una linea tratteggiata, una freccia e lo stereotipo «occurrence».



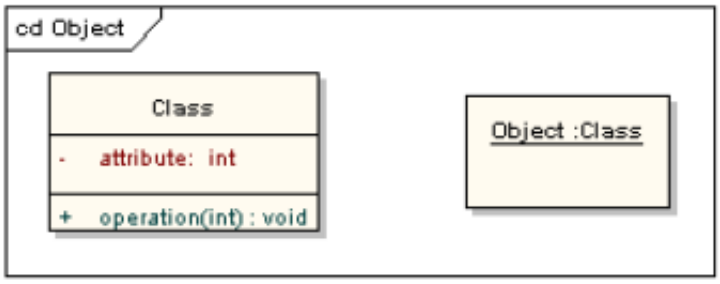
0.8 Diagramma degli oggetti

0.8.1 Diagrammi degli oggetti

Un diagramma degli oggetti può essere considerato uno speciale caso di diagramma delle classi. Essi sono un sottoinsieme di elementi di un diagramma delle classi che enfatizzano le relazioni tra le istanze di una classe in uno stesso istante di tempo. Sono utili per capire i diagrammi delle classi. Non mostrano niente di architetturalmente diverso dai diagrammi delle classi ma riflettono molteplicità e ruoli.

0.8.2 Classi e oggetti

Il seguente diagramma mostra le differenze visive tra i due. Si noti che la classe è costituita da 3 parti, nome, attributi e metodi; di default invece gli oggetti non hanno compartimenti. Il nome mostrato è differente, infatti gli oggetti hanno il nome sottolineato e potrebbero mostrare il nome del loro classificatore.

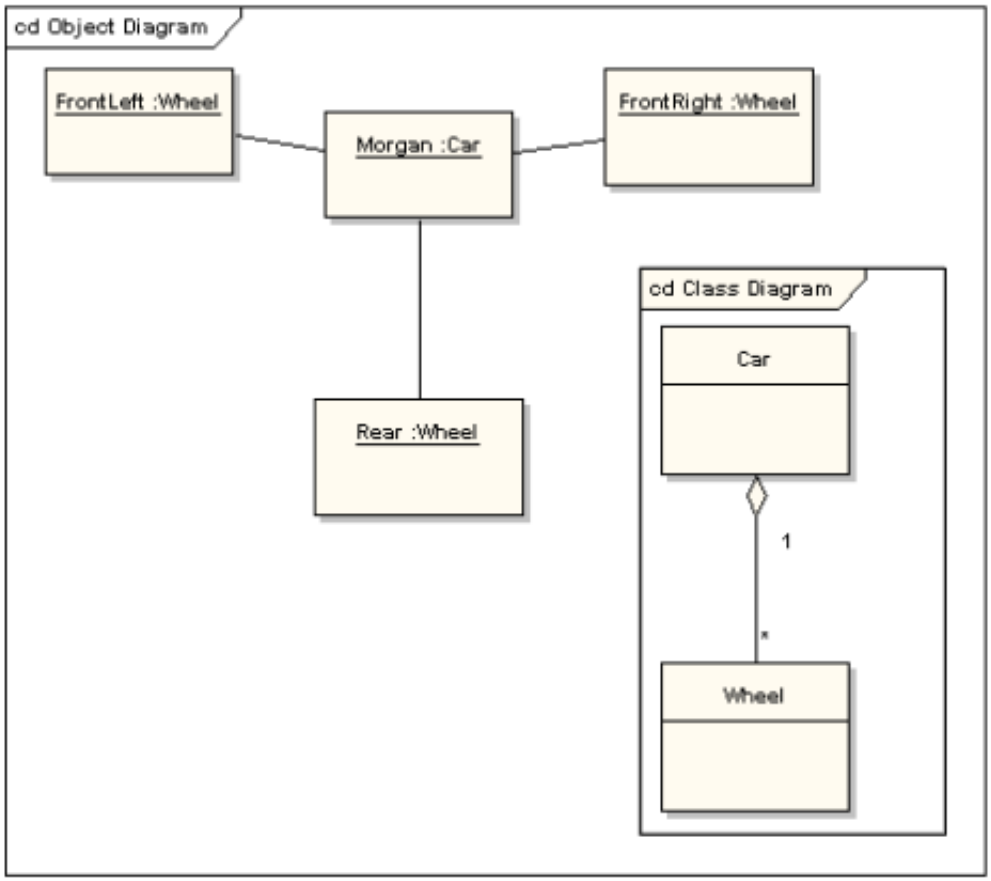


0.8.3 Stato di run-time

Un elemento classificatore può avere un qualunque numero di operazioni e attrivuti. Questi non sono mostrati nell'istanza oggetto. È tuttavia possibile definire lo stato a tempo di esecuzione degli oggetti, mostrando l'insieme di valori degli attributi in una particolare istanza.

0.8.4 Esempio di diagrammi di classi e oggetti

Il seguente diagramma mostra un diagramma degli oggetti con le sue classi di definizioni dentro, e ci mostra il modo in cui un diagramma degli oggetti può essere usato per testare molteplicità di assegbamenti nel diagramma delle classi. La classe car ha una molteplicità 1-n verso la classe wheel, ma se una molteplicità 1-4 fosse stata scelta, allora non sarebbe stato possibile fare una macchina con 3 ruote come quella sotto:



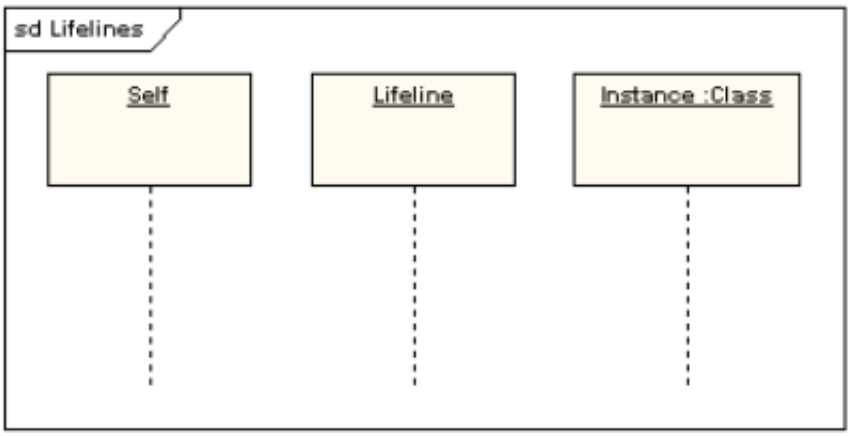
0.9 Diagramma di sequenza

0.9.1 Diagrammi di sequenza

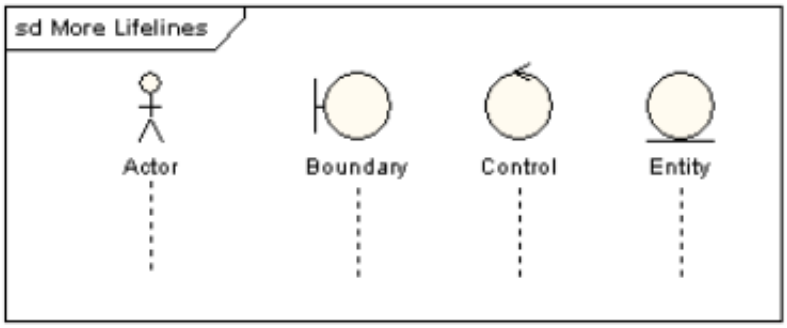
Un diagramma di sequenza è un diagramma di interazioni che mostrano gli oggetti come linee di vita che vanno verso il basso, e le loro interazioni nel tempo rappresentate come messaggi disegnati con frecce dalla linea sorgente dino alla linea obiettivo. I diagrammi di sequenza sono ottimi per mostrare quali oggetti comunicano tra loro e che messaggi si scambiano durante queste comunicazioni. I diagrammi di sequenza non sono pensati per procedure logiche complesse.

0.9.2 Linee di vita

Una linea di vita rappresenta un partecipante individuale nel diagramma. Una linea di vita avrà sempre un rettangolo contenente il nome dell'oggetto. Se il nome è 'self' allora questo indica che la linea di vita rappresenta il classificatore che possiede il diagramma di sequenza.

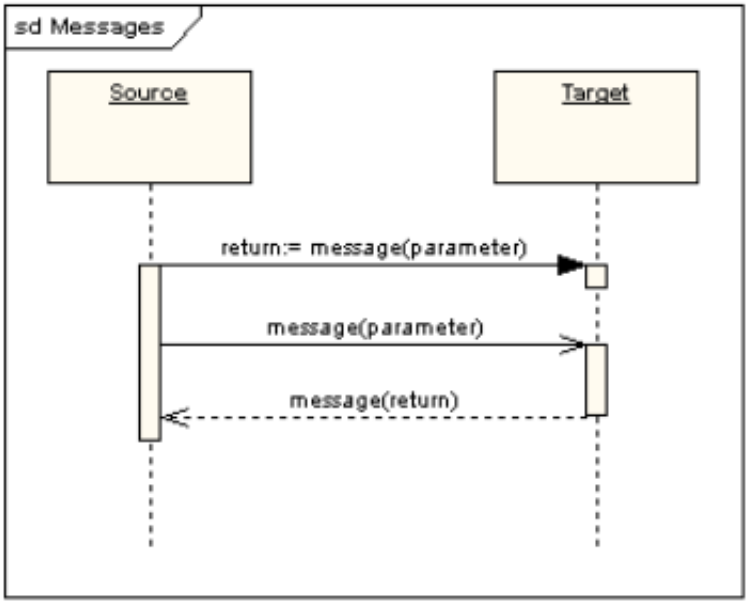


A volte i diagrammi di sequenza possono avere linee di vita con attori invece che rettangoli. Questo è di solito il caso in cui il diagramma di sequenza narra la sequenza di un caso d'uso.



0.9.3 Messaggi

I messaggi sono formati da frecce. Possono essere completi, trovato/perso, sincroni o asincroni, chiamate o segnali. Nel diagramma seguente il primo messaggio è sincrono, denotato da una freccia nera, completato da un messaggio di ritorno implicito; il secondo invece è asincrono, denotato da una freccia vuota e il terzo è il messaggio di ritorno asincrono del secondo, denotato da una freccia tratteggiata.

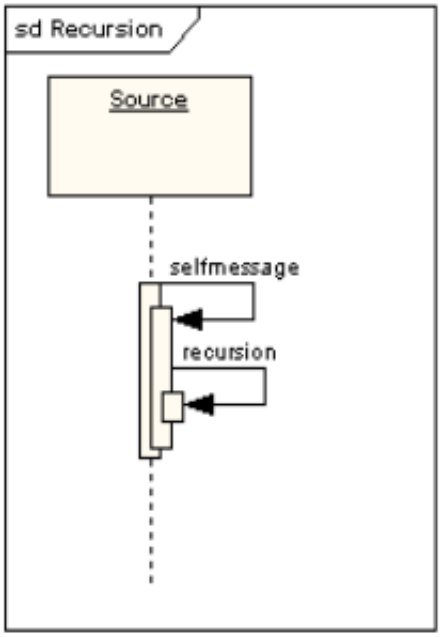


0.9.4 Occorrenza di esecuzione

Un rettangolo sottile che va verso il basso su una linea di vita denota l'occorrenza di esecuzione o attivazione di un focus del controllo. Nel disegno sopra ci sono 3 occorrenze di esecuzione. La prima è la sorgente dei due messaggi inviati e delle due risposte ricevute, mentre la seconda è l'oggetto obiettivo del primo messaggio sincrono e la terza è l'oggetto obiettivo del messaggio asincrono che rimanda una risposta.

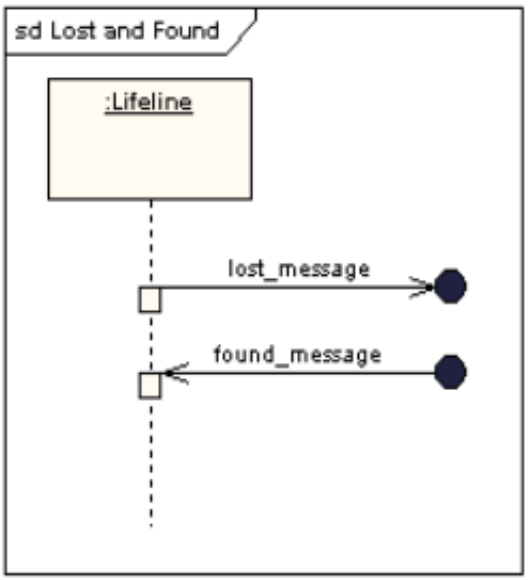
0.9.5 Messaggio self

Un messaggio self può rappresentare una chiamata ricorsiva di un'operazione, un metodo che chiama un altro metodo nello stesso oggetto. È disegnato con una freccia ciclica.



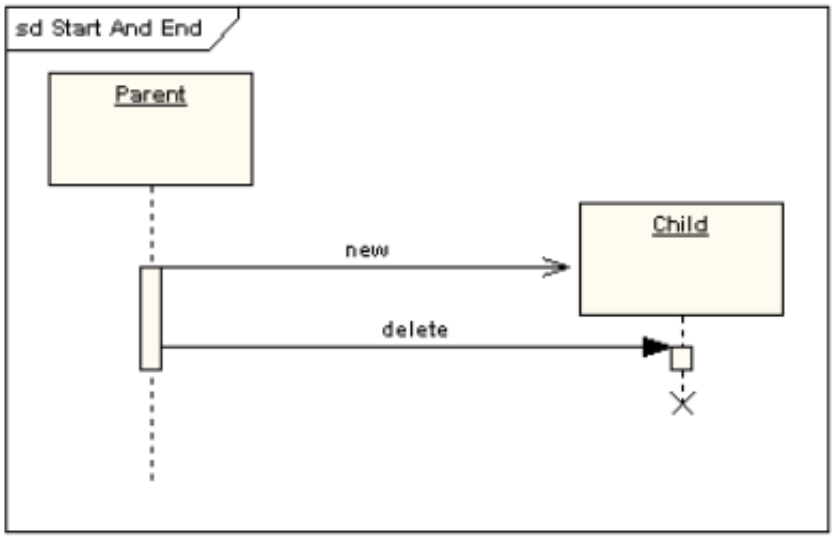
0.9.6 Messaggio di trovato e perso

I messaggi persi sono quei messaggi che sono stati mandati ma non arrivano al destinatario voluto, o che vanno a un destinatario non presente nel diagramma corrente. I messaggi di 'trovato' invece sono quelli che arrivano da un mittente sconosciuto o da un mittente non mostrato nel diagramma corrente. Sono denotati come entranti o uscenti da un elemento endpoint.



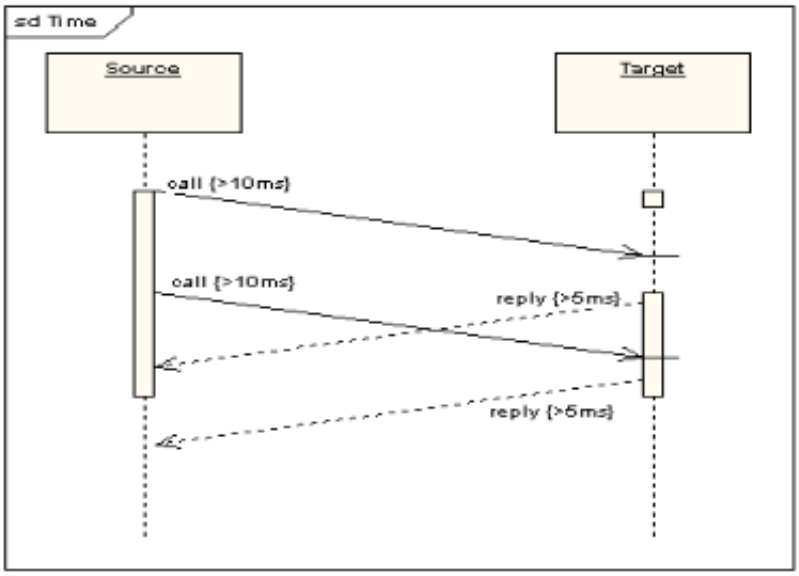
0.9.7 Inizio e fine di linea di vita

Una linea di vita può essere creata o distrutta durante un periodo di tempo rappresentato dal diagramma di sequenza. In quest'ultimo caso, la linea di vita è terminata da un simbolo di stop, rappresentato da una croce. Nel primo caso invece, la testa simbolo della linea di vita è disegnata a un livello più basso della pagina rispetto alle altre, come mostrato qui di seguito.



0.9.8 Durata e vincoli temporali

Di default, un messaggio è mostrato come una linea orizzontale. Dal momento che la linea di vita rappresenta il passare del tempo verso il basso, quando si modella un sistema a tempo reale o un processo aziendale limitato nel tempo, può essere importante considerare la lunghezza del tempo che ogni azione da svolgere richiede. Impostando un vincolo di durata per un messaggio, questo sarà rappresentato da una freccia obliqua:

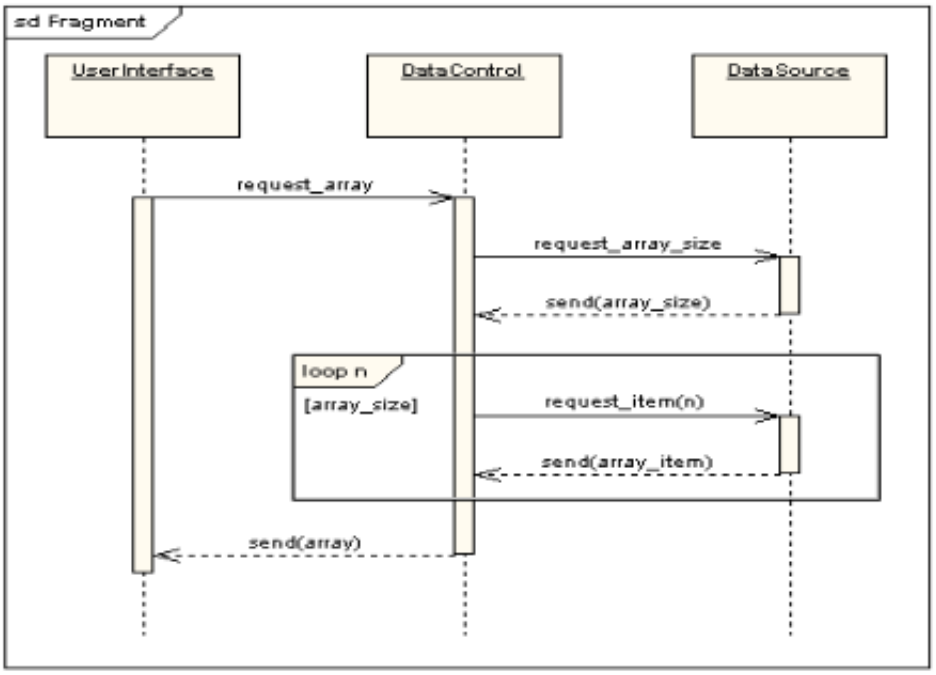


0.9.9 Frammenti combinati

È stato detto prima che il diagramma delle sequenze non è pensato per procedure logiche complesse. Mentre questo è il caso in cui c'è un numero di meccanismi che ammette l'aggiunta di una procedura logica al diagramma e che rientrano nella voce di frammenti combinati. Essi sono una o più sequenze dentro un frame che sono eseguite sotto specifiche circostanze. I frammenti disponibili sono:

- Frammenti alternativi (alt) che sono in pratica if..then..else
- Frammenti opzionali (opt) che modellano lo switch.
- Frammenti di break, che modellano una sequenza alternativa di eventi che è processata al posto di tutto il resto del diagramma
- Frammenti paralleli (par) che modellano processi concorrenti
- Frammenti di Sequenza debole (seq) che rappresentano un insieme di sequenze per cui tutti i messaggi devono essere processati in un precedente segmento prima di far partire il successivo, ma che non impone nessuna sequenza dentro i segmenti con i messaggi che non condividono una linea di vita
- Frammenti di sequenza rigorosi (strict) racchiudono una serie di messaggi che devono essere processati in un dato ordine
- Frammenti negativi (neg) racchiudono serie di messaggi non validi
- Frammenti critici, racchiudono una sezione critica
- Frammenti di ignoro, stabiliscono se un messaggio può apparire non interessante nel contesto corrente
- Frammenti di considerazione, che sono l'opposto dei frammenti di ignoro
- Frammenti assertivi (assert) che stabiliscono che ogni sequenza non mostrata come operando di asserzione non è valida

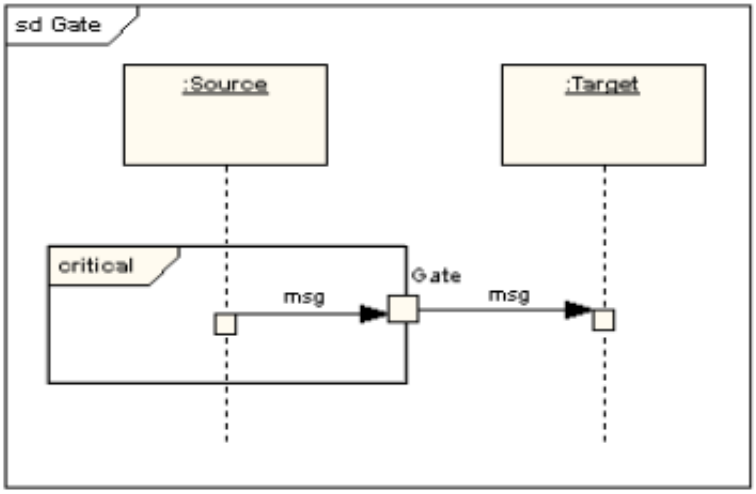
- Frammenti Loop (loop) racchiudono una serie di messaggi ripetuti.



C'è anche un'interazione di occorrenza, che è simile al frammento combinato. Un'interazione di occorrenza è un riferimento a un altro diagramma con la parola 'ref' nell'angolo sinistro superiore, e ha lo stesso nome del diagramma di riferimento mostrato nel mezzo del frame.

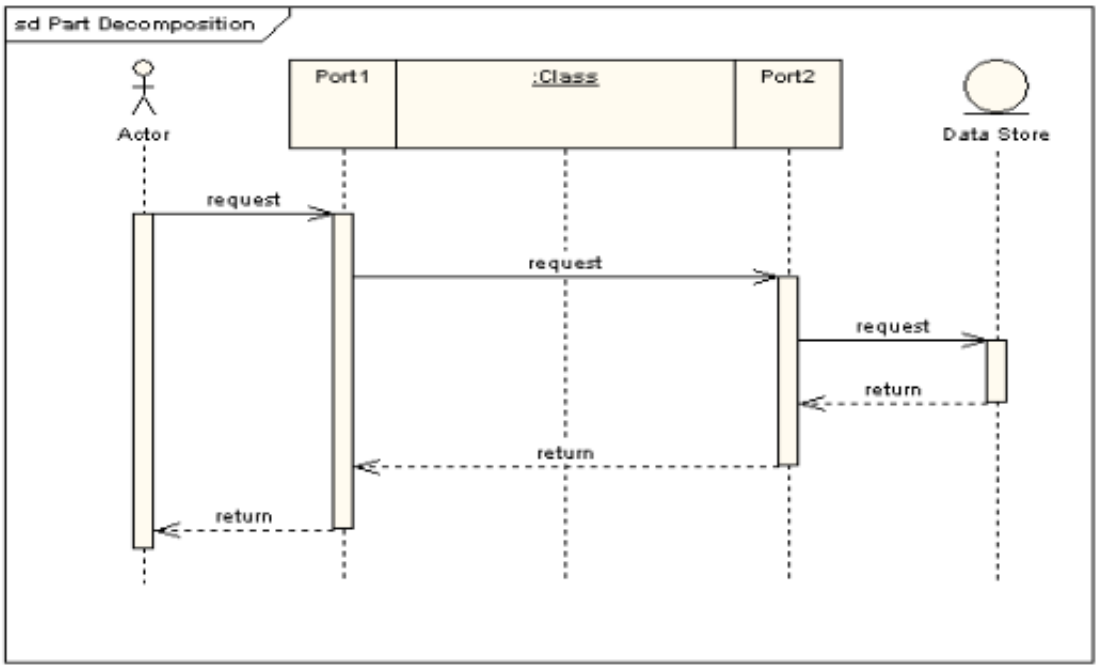
0.9.10 Cancello

Un cancello è un punto di connessione tra un messaggio interno a un frammento e un messaggio all'esterno di esso.



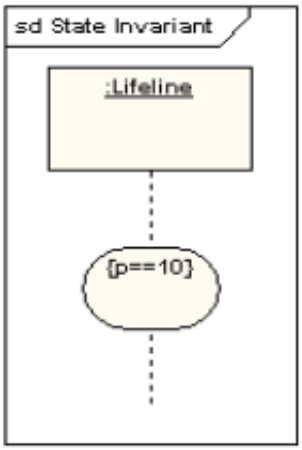
0.9.11 Decomposizione di parti

Un oggetto può avere una o più linee di vita che partono da esso, e questo permette i messaggi interni ad esso di essere scambiati con se stesso nello stesso diagramma.



0.9.12 Stato invariante/continuazioni

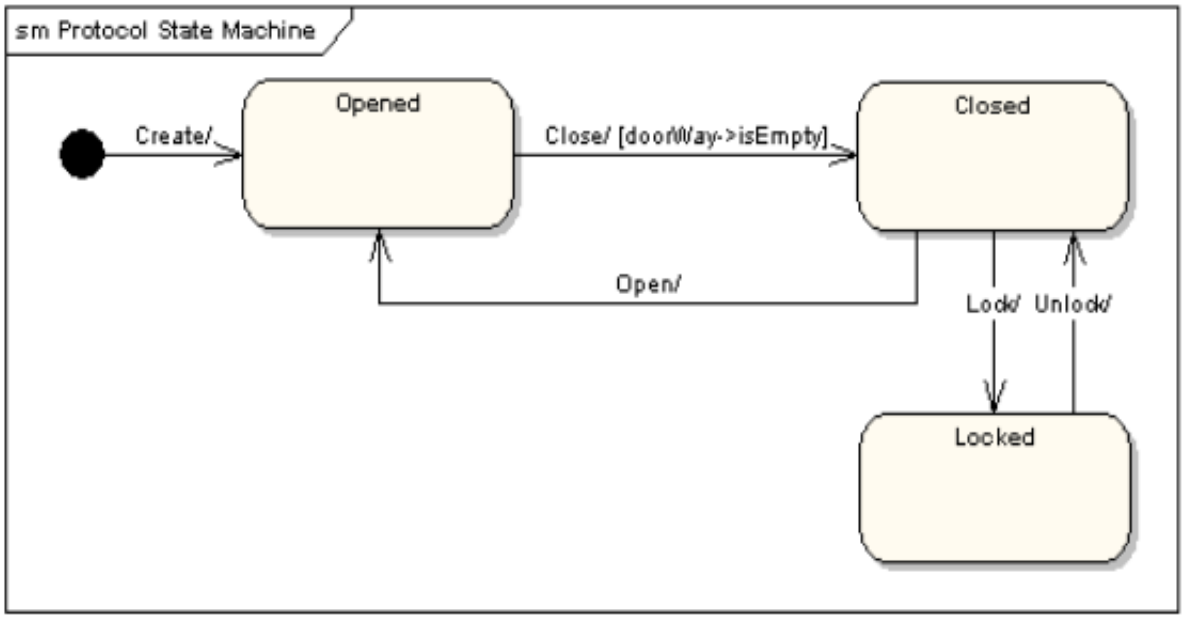
Uno stato invariante è un vincolo piazzato su una linea di vita che deve essere vero a run-time. Disegnato con un rettangolo stondato. La continuazione ha la stessa notazione ma è usato in frammenti combinati e può essere allargato a più linee di vita.



0.10 Diagramma di macchina a stati

0.10.1 Diagrammi di macchina a stati

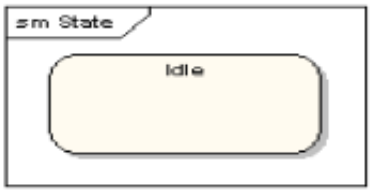
Un diagramma di macchina a stati modella il comportamento di un singolo oggetto, specificando la sequenza degli eventi che un oggetto attraversa durante la sua vita in risposta ad eventi.



La door può essere in tre stati: Opened, Closed o Locked. Essa può rispondere agli eventi Open, Close, Lock e Unlock. Si noti che non tutti gli eventi sono validi in tutti gli stati: per esempio se una Door è Opened non puoi Lockarla fino a che non la chiudi. Si noti anche lo stato di transizione che può avere una guardia condizionale allegata: se la Door è Opened, può solo rispondere all'evento Close anche se la confizione doorWay->isEmpty è vera. La sintassi e le convenzioni usate nel diagramma di macchine a saranno nel seguito discusse.

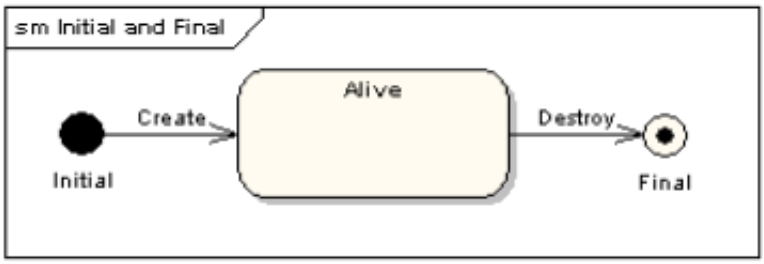
0.10.2 Stati

Uno stato è denotato da un rettangolo stondato con il nome dello stato scritto dentro.



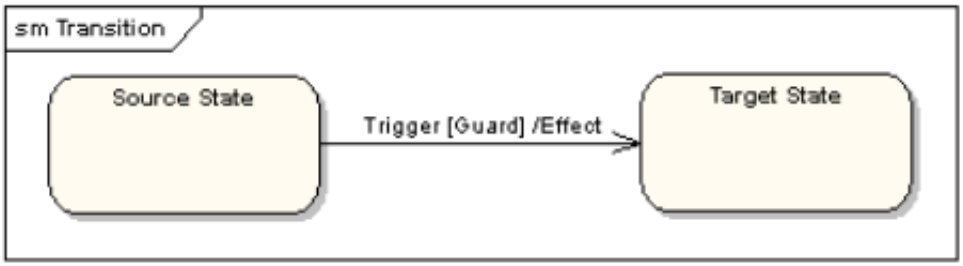
0.10.3 Stato iniziale e finale

Lo stato iniziale è indicato da un cerchio nero e può avere un nome. Lo stato finale è denotato da un cerchio contenente un cerchio nero e anch'esso può avere un nome.



0.10.4 Transizioni

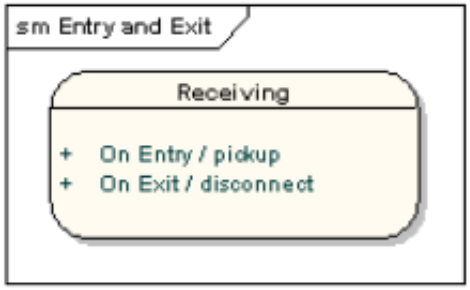
Le transizioni da uno stato al successivo sono denotati da frecce. Una transizione può avere una guardia, un trigger o un effetto, come segue:



Il trigger è la causa della transizione, che potrebbe essere un segnale, un evento o un cambiamento di qualche condizione, o il passaggio del tempo. La guardia è la condizione che deve essere vera per permettere al trigger di causare la transizione. L'effetto è l'azione che sarà invocata direttamente sull'oggetto che possiede lo stato come risultato di una transizione.

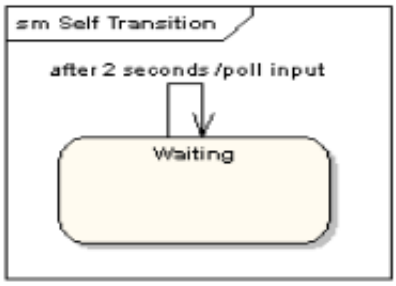
0.10.5 Stati azione

Nell'esempio di sopra l'effetto era associato alla transizione. Ma se lo stato obiettivo ha più transizioni in arrivo ed ognuna ha lo stesso effetto associato a ciascuna di esse, sarebbe meglio associare l'effetto al target piuttosto che alle singole transizioni. Questo può essere fatto definendo un'azione di entrata per lo stato. È pure possibile definire azioni che occorrono durante eventi o azioni che occorrono spesso. È possibile definire un numero di azioni di qualsiasi tipo.



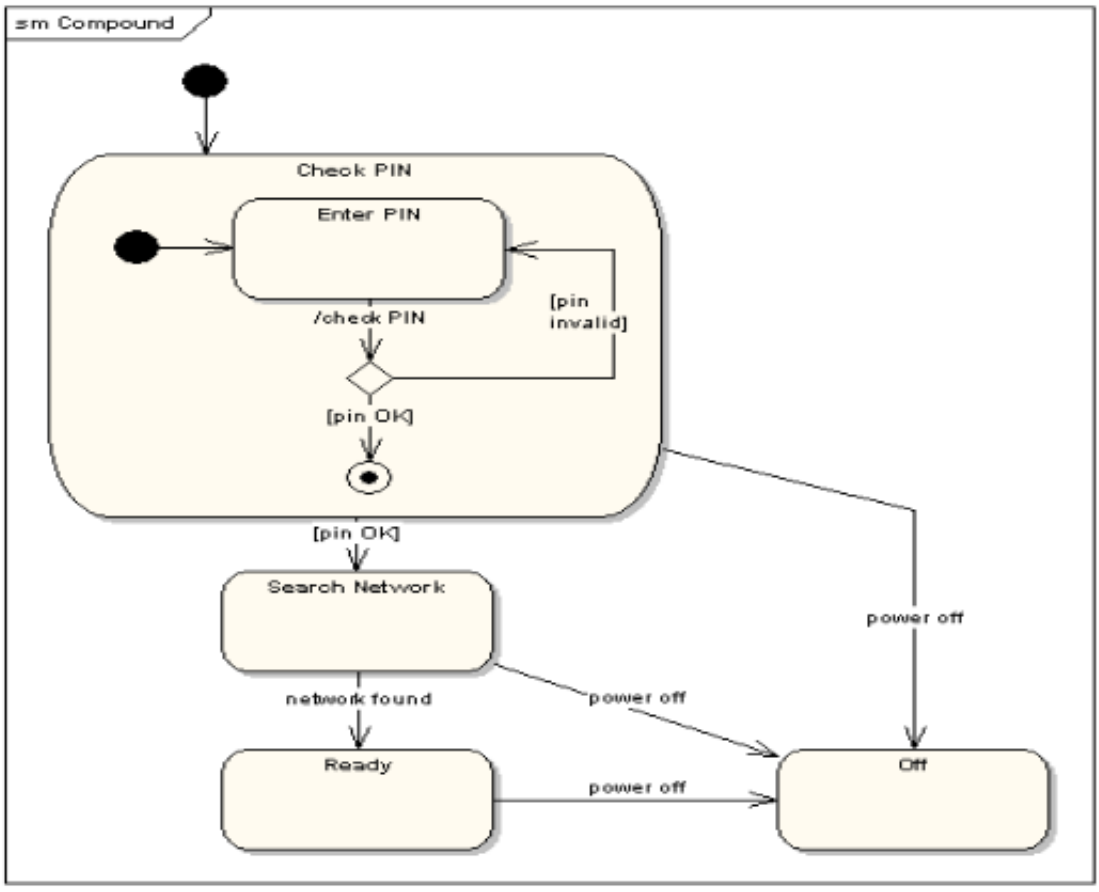
0.10.6 Transizioni self

Uno stato può avere una transizione che torna in se stessa. È molto utile quando un effetto è associato a una transizione.

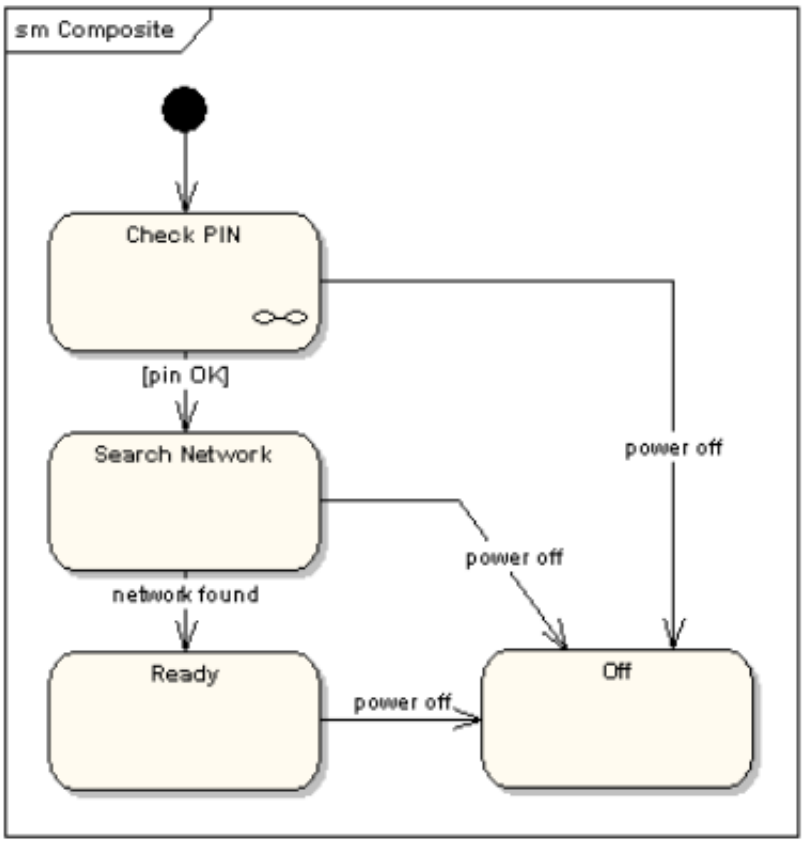


0.10.7 Stati composti

Un diagramma di macchina a stati potrebbe includere sottodiagrammi di macchine a stati:



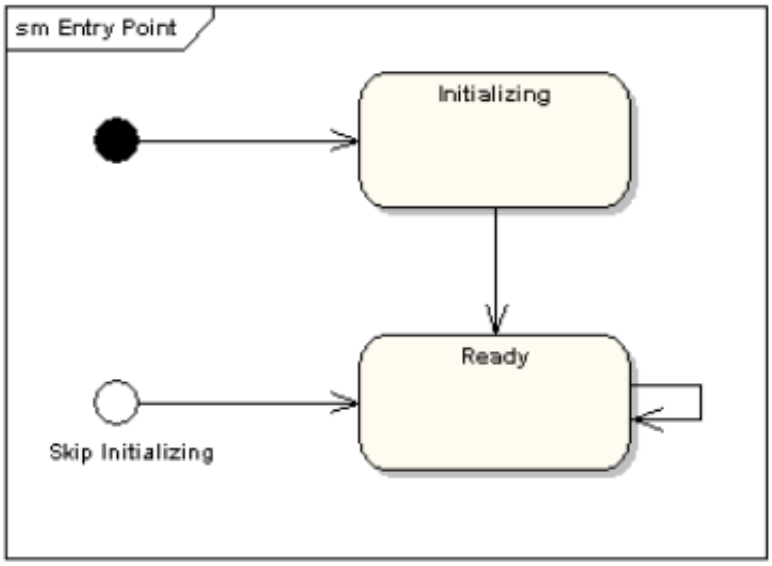
Alternativamente possiamo fare lo stesso in questo modo:



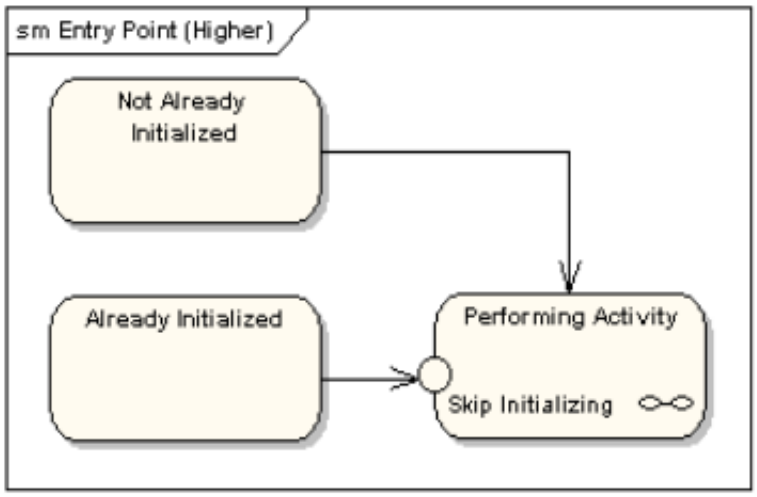
La notazione qui in alto indica che i dettagli della sottomacchina check-PIN sono mostrati in diagrammi distinti.

0.10.8 Punti di ingresso

A volte si potrebbe non voler entrare in una sottomacchina dallo stato iniziale, come nell'esempio seguente; magari non è sempre necessario inizializzare, quindi si potrebbe usare una transizione di tipo Entry Point che permette di entrare già dallo stato Ready.

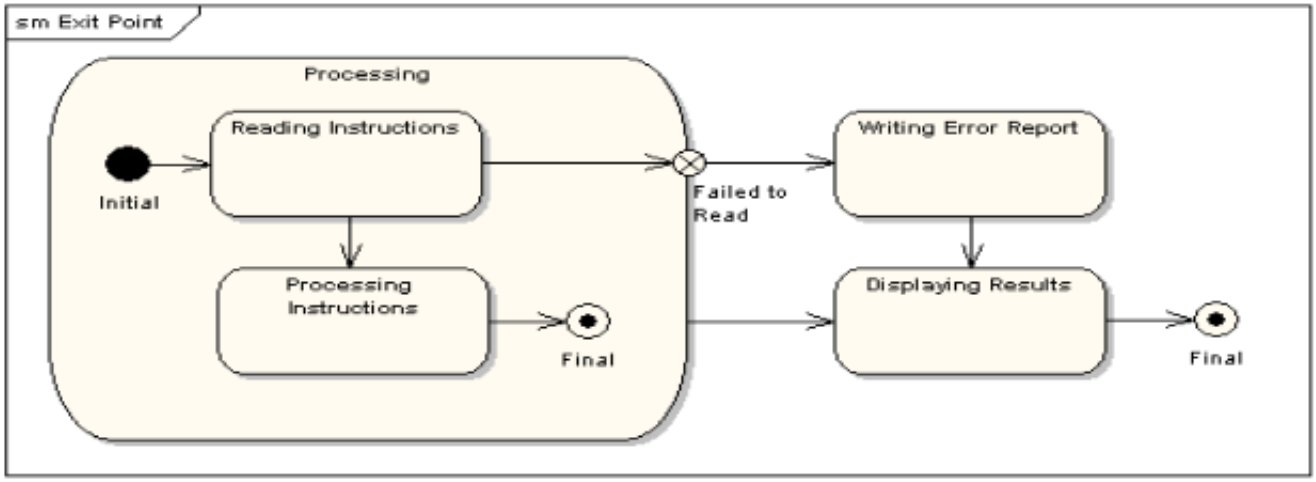


E il livello superiore sarebbe di questo tipo:



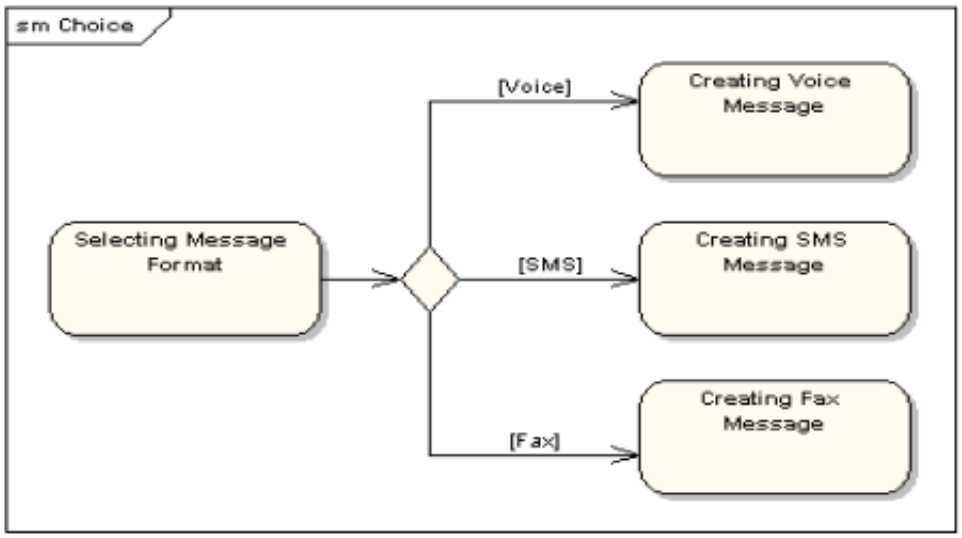
0.10.9 Punti d'uscita

In modo analogo ai punti di ingresso, è possibile averci delle alternative uscite nominate. Il seguente diagramma da' un perfetto esempio dove lo stato eseguito dopo il processo principale dipende da quale percorso si sceglie per uscire da questo.



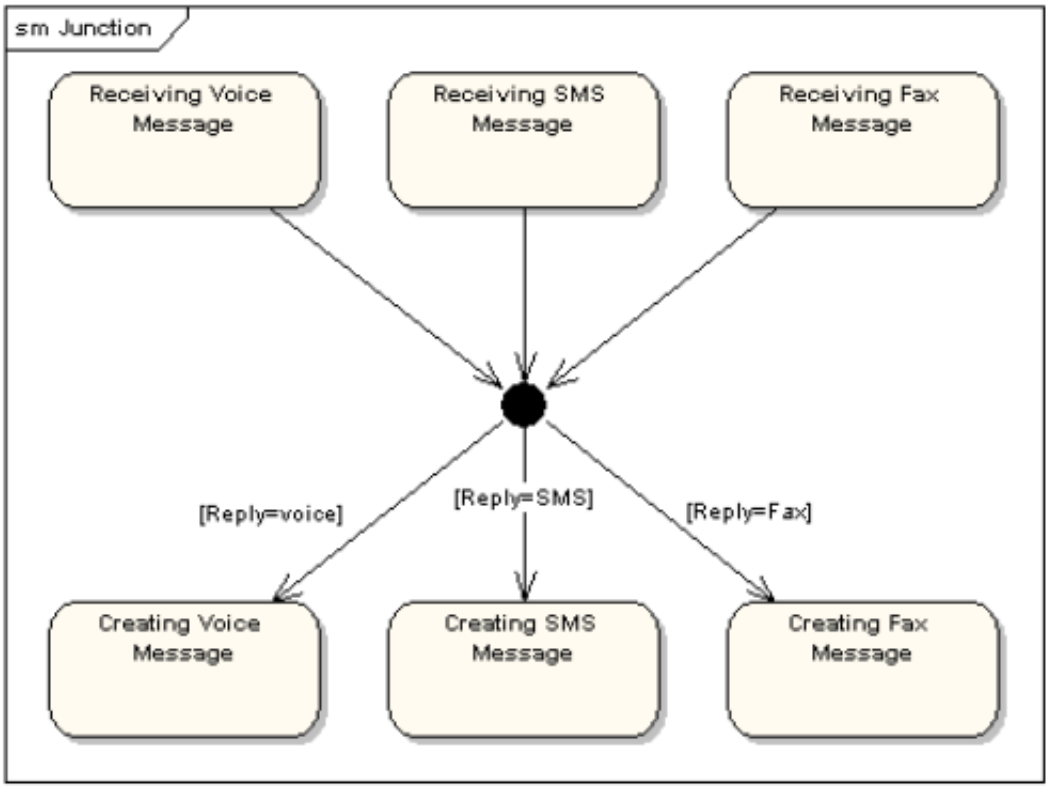
0.10.10 Scelta di pseudo-stato

Una scelta di pseudo-stato è mostrata con un rombo con un ingresso e 2 o più uscite. Il seguente esempio mostra che qualunque sia lo stato in arrivo, dopo la scelta lo pseudo-stato è dipendente dal formato del messaggio selezionato durante l'esecuzione dello stato precedente.



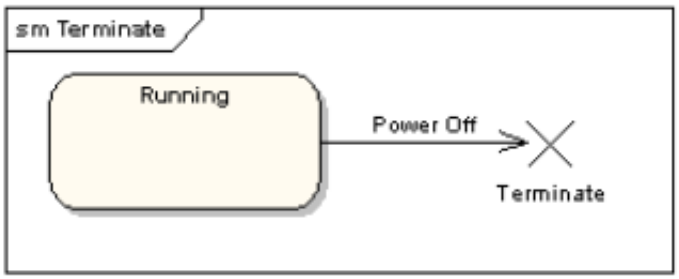
0.10.11 Pseudo stato di giunzione

Sono usati per collegare iterazioni multiple. Una singola giunzione può avere uno o più ingressi e una guardia può essere applicata a ciascuna transizione. Le giunzioni sono libere da semantica; una giunzione che divide una transizione in ingresso in più transizioni in uscita realizza un controllo condizionale statico rispetto allo pseudo-stato di scelta, che realizza un controllo condizionale dinamico.



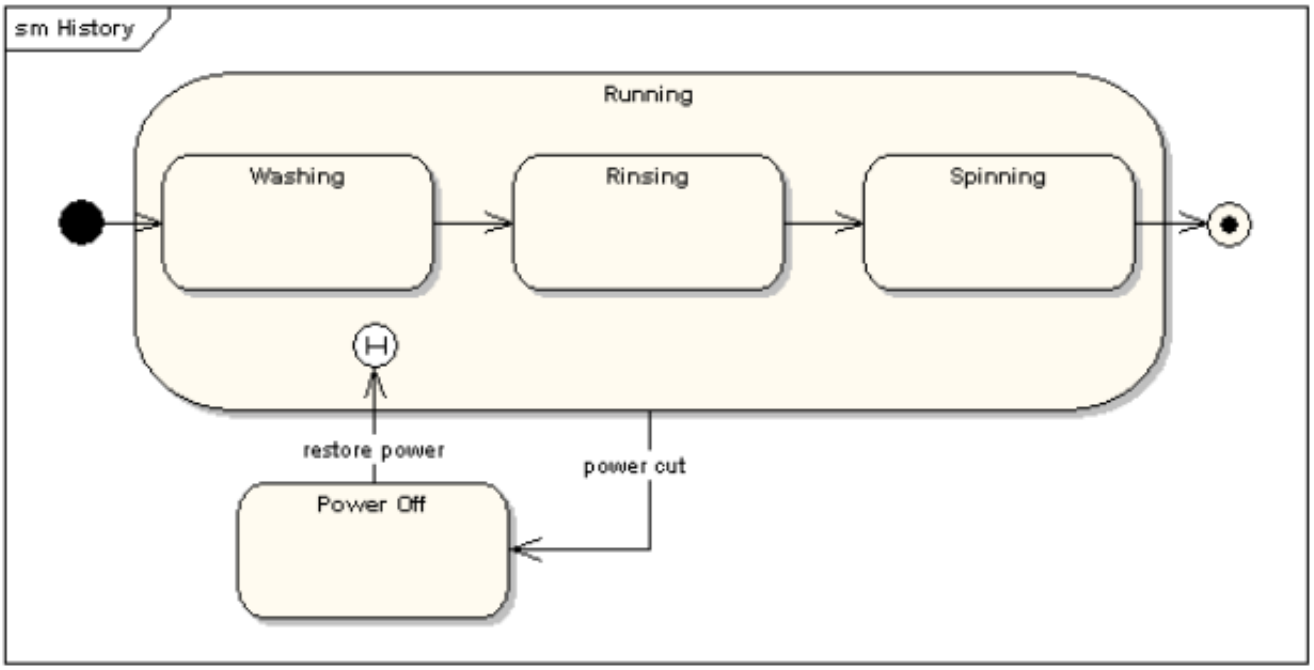
0.10.12 Pseudostato di terminazione

Entrare in uno pseudo-stato di terminazione indica che la linea di vita dello stato della macchina è finito. È denotato da una croce.



0.10.13 History states

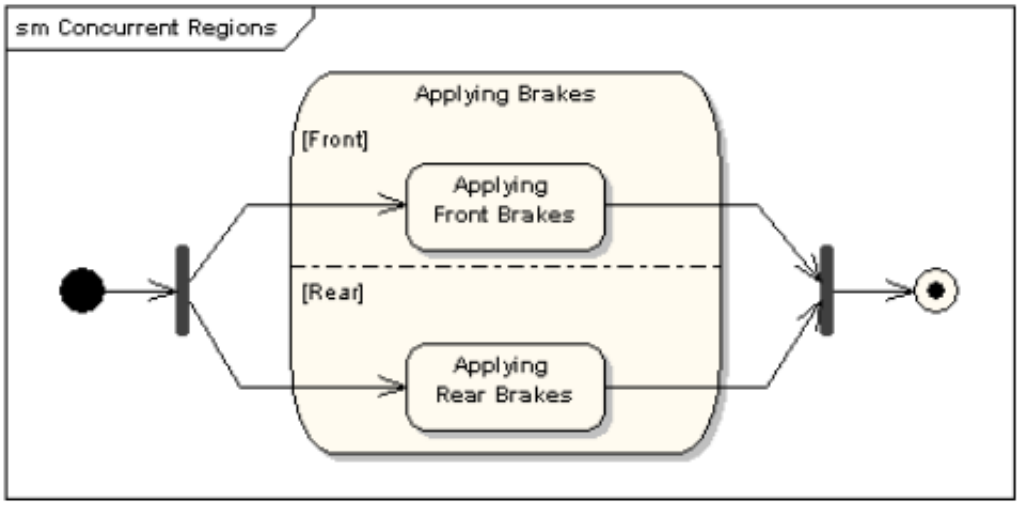
Un history state è usato per ricordare il precedente stato di una macchina a stati quando è statat interrotta. Il seguente esempio mostra un suo utilizzo: (lavatrice)



In questa macchina a stati, mentre la lavatrice è in esecuzione c'è una progressione da lavando a risciacquando a centrifugare. Se c'è un balzo di corrente, la lavatrice si blocca e andrà nello stato di spenta. Quando la corrente si riattiva, la lavatrice entrerà nell'History State e riprenderà il lavoro da dove era interrotto.

0.10.14 Regioni concorrenti

Uno stato può essere diviso in regioni contenenti sottostati che esistono e sono eseguiti concorrentemente. L'esempio sotto mostra che all'interno dello stato Applying Brakes, i freni anteriori e posteriori opereranno contemporaneamente ed indipendentemente. Si noti l'utilizzo di una fork e di una join piuttosto che di una choice e una merge. Questi simboli sono usati per sincronizzare i thread concorrenti.



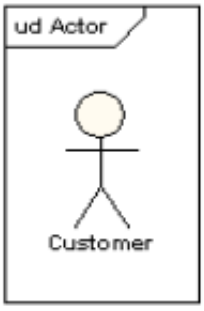
0.11 Diagramma dei casi d'uso

0.11.1 Modello dei casi d'uso

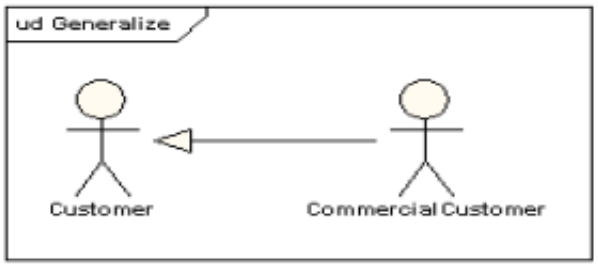
Il modello dei casi d'uso cattura i requisiti di un sistema. I casi d'uso sono un mezzo per comunicare con gli utenti e le altre parti interessate su ciò che il sistema è destinato a fare.

0.11.2 Attori

Un diagramma dei casi d'uso mostra le interazioni tra il sistema e entità esterne ad esso. Queste entità sono riferite come Attori. Gli attori rappresentano i ruoli che potrebbero includere utenti umani, hardware esterni o altri sistemi. Un attore è solitamente disegnato con un omino stilizzato o alternativamente come una classe con la parola chiave «actor».

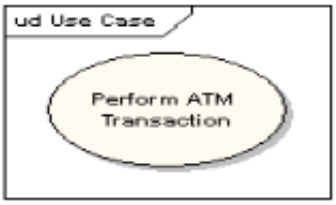


Un attore può generalizzare altri attori come più dettagliati:

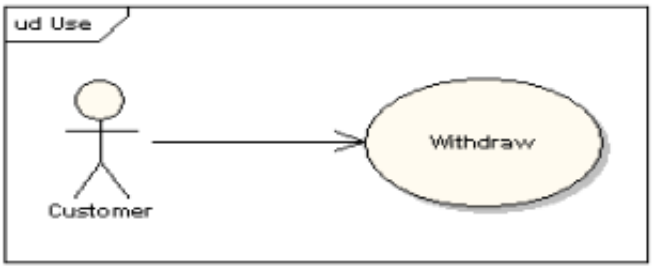


0.11.3 Casi d'uso

Un caso d'uso è una singola unità di lavoro significativo. Fornisce una vista ad alto livello di un comportamento osservabile a qualcuno o qualcosa all'esterno del sistema. La notazione è un ellisse:



La notazione per usare un caso d'uso è collegarlo con una linea con una freccia opzionalmente, mostrando la direzione del controllo.



0.11.4 Definizione di un caso d'uso

Un caso d'uso tipicamente include:

- Nome e descrizione
- Requisiti
- Vincoli
- Scenari
- Diagramma degli scenari
- Informazioni aggiuntive

0.11.5 Nome e descrizione

Un caso d'uso normalmente ha un nome che è un verbo o frase e ha una breve descrizione testuale.

0.11.6 Requisiti

I requisiti definiscono le funzionalità formali che un caso d'uso deve fornire all'utente finale. Corrispondono ai requisiti funzionali trovati nelle metodologie strutturali. Un requisito è un contratto o una promessa per cui un caso d'uso svolgerà azioni o fornirà valore al sistema.

0.11.7 Vincoli

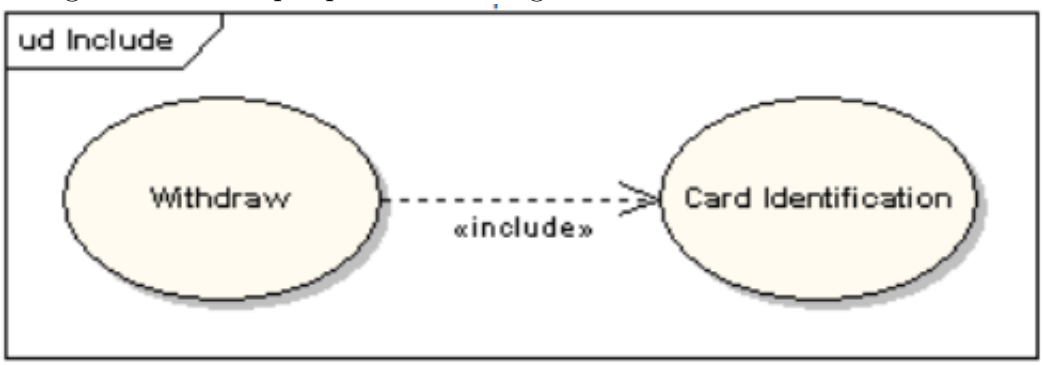
Un vincolo è una condizione o restrizione che un caso d'uso considera, e include pre e post condizioni e condizioni di invarianza. Una precondizione specifica una condizione che deve essere affrontata prima che il caso d'uso possa procedere, mentre una post condizione è lo stato cambiato dopo l'esecuzione di un caso d'uso. Una condizione di invarianza specifica le condizioni che sono vere durante tutta l'esecuzione del caso d'uso.

0.11.8 Scenari

Uno scenario è una descrizione formale del flusso di eventi che occorrono durante l'esecuzione di un'istanza di caso d'uso. Definisce la sequenza specifica degli eventi tra il sistema e gli attori esterni. È normalmente descritta in forma testuale e corrisponde alla rappresentazione testuale del diagramma di sequenza.

0.11.9 Includere i casi d'uso

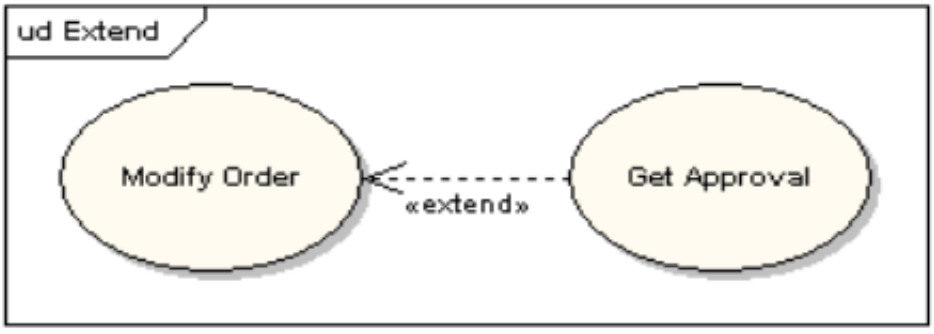
I casi d'uso possono contenere funzionalità di altri casi d'uso come parte del loro normale processo. In generale è assunto che un caso incluso è chiamato ogni volta che il caso base è eseguito. Un esempio può essere il seguente:



I casi d'uso possono essere inclusi da uno o più casi d'uso, aiutando a ridurre la ridondanza di casi d'uso uguali inclusi molteplici volte.

0.11.10 Estendere i casi d'uso

Un caso d'uso può anche estendere il comportamento di un altro; questo è usato tipicamente in circostanze eccezionali. Per esempio, prima di modellare una particolare richiesta di un cliente, un utente deve essere approvato da un'autorità più alta, quindi il caso d'uso 'Get Approval' può opzionalmente estendere il caso d'uso regolare 'Modify Order'.



0.11.11 Limiti di sistema

È normale vedere i casi d'uso all'interno del sistema e gli attori al di fuori.