Nei diagrammi di contesto l'ENVIROMENT (contesto) comprende tutte le entità esterne al sw su cui non abbiamo controllo —> ci deve essere chiaro cosa è dentro e cosa è fuori.
I modelli di contesto vengono utilizzati per evidenziare i confini del mio sw e quali sono le entità esterne

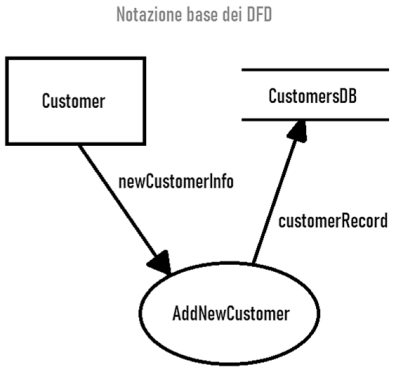
I diagrammi di contesto adattano una visione esterna, servono a mostrare i confini del sw.
I DFD di livello 0 hanno un'entità singola (il sistema sw) e tutto il resto è esterno —> quindi i DFD di liv 0 (i DFD sono modelli comportamentali) sono modelli di contesto.

DFD – Data Flow Diagram

Sono modelli comportamentali

I DFD servono a identificare i processi di elaborazione/trasformazione dei dati elaborati in un sistema software. Nei DFD vengono specificati dati in input e dati prodotti in output (ossia flussi in ingresso e flussi in uscita). Un DFD può essere costruito a vari livelli gerarchici: un singolo processo che è caratterizzato da precisi input e output può essere scomposto nei suoi sotto processi, che globalmente avranno gli stessi input e output (eventualmente descritti ad un livello di dettaglio più alto – corrispondenza dei flussi). Questo processo di scomposizione è detto scomposizione funzionale.

Notazione:

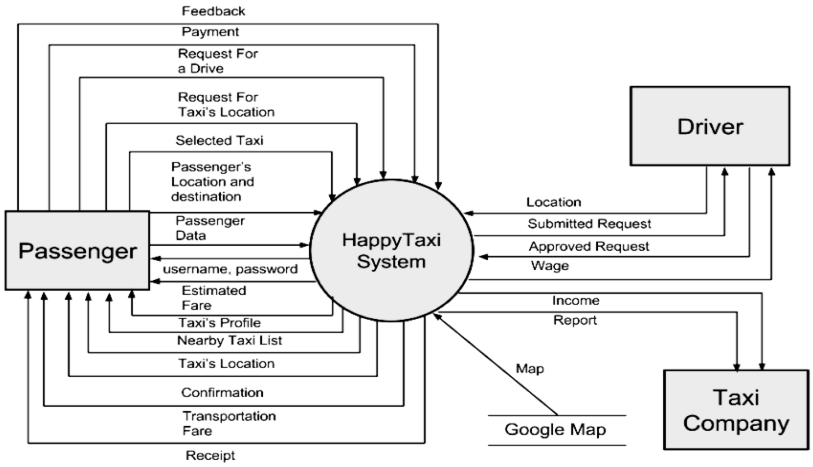


- I **dati** vengono associati ai flussi e devono venir descritti mediante nomi che indichino dati (staticità) e non processi (dinamicità).
- I **flussi** vengono rappresentati dalle frecce (con estremità "piena"). Vengono anche chiamati pipeline.
- I **processi** vengono associati alle elissi e devono venir descritti mediante nomi che indichino processi/attività. (dinamicità e non staticità).
- Gli **archivi** di dati possono venir indicati da linee parallele orizzontali.
- Le **entità esterne** possono venir rappresentate alle estremità delle frecce entranti o uscenti del diagramma, utilizzando rappresentazioni iconiche, rettangoli o semplicemente un nome che le denota.

Diagramma di contesto e DFD di livello 0

Un diagramma DFD di livello 0 corrisponde ad un **diagramma di contesto**. Nel livello 0 il sistema è rappresentato con un singolo nodo (processo) e con i flussi di dati che vengono scambiati con unità esterne al sistema. Quindi: c'è una solta entità che rappresenta il processo centrale dello scope del sistema. Mostra cosa il sistema riceve e cosa restituisce.

Esempio diagramma di contesto (DFD livello 0):



Non vanno confusi i DFD con i Flow Chart. I DFD descrivono le trasformazioni sui dati e il relativo flusso di trasformazione da input a output. I Flow chart (Diagrammi di flusso) invece descrivono il controllo, quali operazioni vengono fatte prima e quali dopo, e quali test eseguire per scegliere fra flussi diversi.

I DFD non ci spiegano come avviene la sincronizzazione, ovvero A —> B non sappiamo se vuol dire A controlla B, A produce e B consuma, ecc.

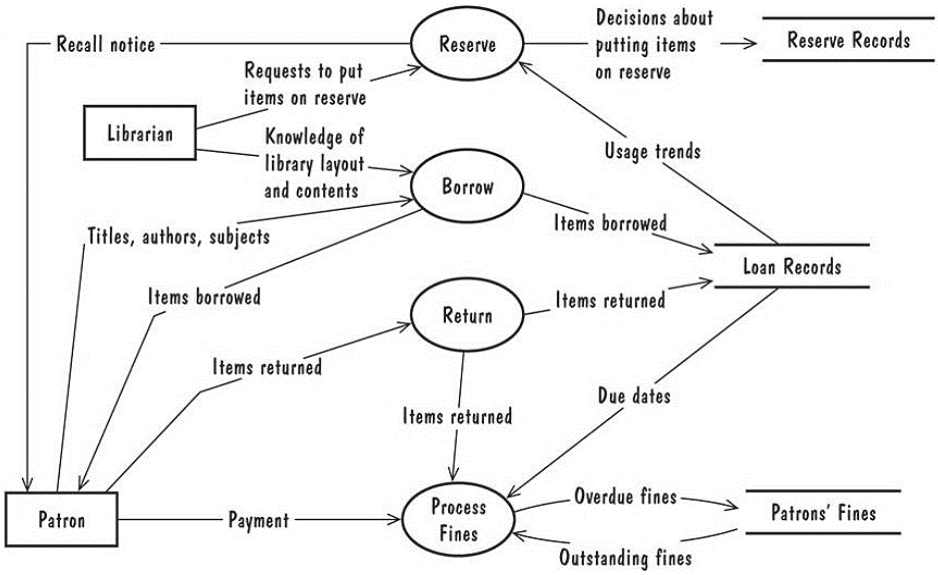
Linee guida

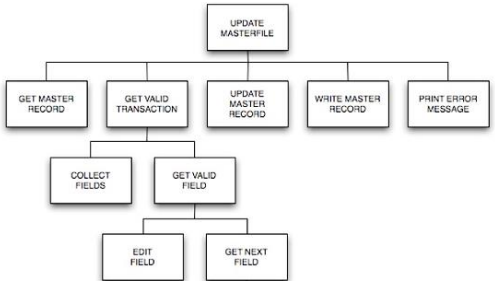
Linee guida per la stesura di un DFD

- **Ignorare** condizioni e operazioni di inizializzazione, terminazione, gestione degli errori. (si pensi quindi alla situazione stabile, non al transitorio, inizializzazione, all'avvio, alla terminazione).
- **Ignorare** il flusso (la logica) di controllo e la sincronizzazione delle fasi.
- **Ignorare** i dati di controllo (è un dettaglio implementativo; eccessivo).
- **Ignorare** il funzionamento interno di un processo (considerarlo una black box) ma modellarne solo gli aspetti funzionali ed i relativi flussi di I/O.
- **VERIFICARE** la coerenza di dati e risultati e viceversa.

Esempio DFD sistema bibliotecario:

(NB: le frecce dovrebbero essere "piene")



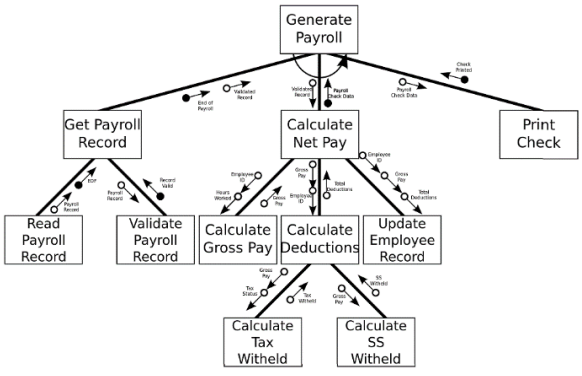


Schema HIPO: Hierachical Input Process Output

Ogni funzione è rappresentata da un box rettangolare e può essere descritta più a fondo in un diagramma HIPO: ogni funzione è strutturata in sotto funzioni, livello dopo livello. È tipico disporre a sinistra i moduli input, in centro di processing e a destra di output.

Structure chart (SC)

Un diagramma strutturale (SC) è un diagramma che mostra la suddivisione di un sistema ai suoi livelli gestibili più bassi e lo scambio di dati. Ogni modulo è rappresentato da una casella, che contiene il nome del modulo. La struttura ad albero visualizza le relazioni tra i moduli.



Notazione:

Le reti di Petri sono modelli COMPORTAMENTALI e servono a superare la barriera dei modelli A STATI per sistemi complessi, infatti se il sistema è grande è impensabile creare 2^n stati per mappare tutte i singoli eventi possibili all'interno del programma sw.

Reti di Petri

Le reti di Petri servono a rappresentare la dinamica di un sistema in cui risulta complesso utilizzare gli automi, ed in particolare un unico stato globale (negli automi la situazione specifica in cui si trova l'intero sistema viene rappresentata da un'unica entità astratta detta stato).

Nelle reti di Petri, lo stato viene rappresentato suddiviso/distribuito nelle sue componenti significative, ognuna delle quali è rappresentata da un **posto**. La dinamica del sistema viene descritta con granularità fine, attraverso passaggi di stato (transizioni) riferiti solo ad una parte del sistema (singoli posti o insiemi di posti). In tal modo risulta possibile rappresentare più processi indipendenti tra loro, che però possono caratterizzarsi per varie forme di sincronizzazione. Graficamente, una RdP è rappresentabile come un grafo bipartito.

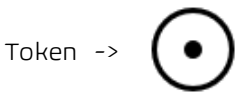
Una rete di Petri (RdP) è composta da quattro elementi fondamentali:

- Un insieme di **posti P**, rappresentati graficamente da un cerchietto.
- Un insieme di **transizioni T**, rappresentati graficamente da una barretta:

- Una **funzione di input** I (I:P -> N) che associa una collezione di posti (posti di input) ad una transazione. I è rappresentata graficamente da un arco orientato da uno o più posti ad una transizione.

- Una **funzione di output** O (O:T -> P) che mappa una transizione in una collezione di posti chiamati posti di output. O è rappresentata graficamente da un arco orientato da una transazione a uno o più posti.

- Un'altra primitiva delle RdP è il **token** rappresentato graficamente da un puntino (interno ad un posto).



- Una **marcatura** è un assegnamento di token sui posti della rete: zero, uno o più token per ciascun posto della rete. Serve a rappresentare lo stato corrente del sistema.

I posti rappresentano stati parziali; la marcatura ci permette di identificare lo stato corrente del sistema, che risulta quindi distribuito, scomposto in stati parziali. Le funzioni di input e di output associati ad una specifica transizione rappresentano rispettivamente i posti da cui è possibile attivare la transizione ed i posti a cui si transita al completamento della transizione. I token sono

astrazioni interpretabili in diversi modi in base alla specifica situazione: possono identificare la quantità di risorse presenti in un posto, oppure se una condizione (associata al posto) è o meno verificata (in un determinato momento), contatori, il numero di volte in cui un posto è stato usato etc.

L'**evoluzione** di una RdP è condizionata dal numero e dalla distribuzione di token nella rete. L'evoluzione è caratterizzata dallo scatto delle transizioni (che rappresenta un passaggio da uno stato del sistema ad uno stato successivo). Una transizione scatta "consumando" dei token da tutti i suoi posti di input, e "mettendone" altri in tutti i suoi posti di output. Le RdP sono quindi event-driven.

L'evoluzione della rete rappresenta l'evoluzione del sistema, da una situazione ad una successiva, ciascuna modellata da più stati parziali (uno stato globale distribuito).

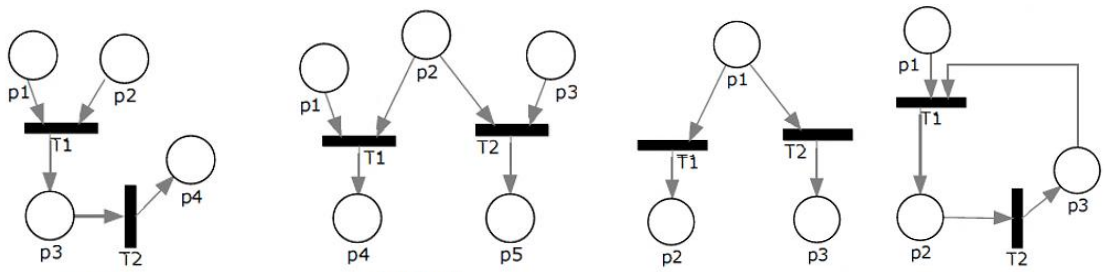
Lo scatto della transizione avviene se si verificano le seguenti condizioni:

- Una transizione è **abilitata allo scatto** se tutti i posti in input contengono token da prelevare. Allo scatto verranno prelevati token dai posti in input alla transizione stessa.
- Una sola transizione per volta può scattare; ciò per evitare che due transizioni abilitate prelevino lo stesso token. La transizione che è scatta è scelta in modo non deterministico.

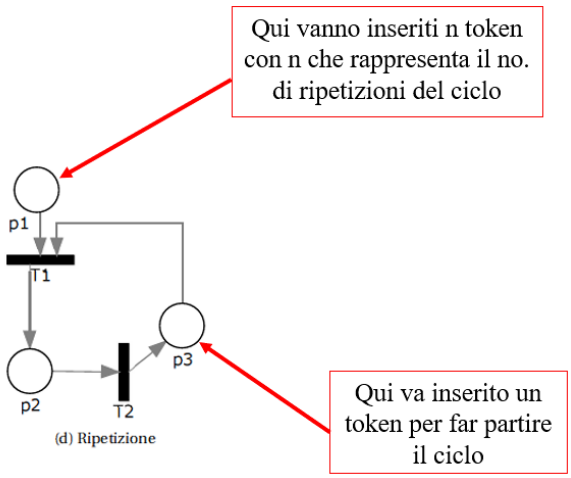
In una visione più generale, un arco può avere associato un peso (positivo). Tale peso se associato ad un arco che connette un posto di input ad una transizione, specifica il numero minimo di token necessari affinché il posto in esame possa dare il proprio contributo ad abilitare la transizione.

Il peso specifica anche quanti token saranno tolti dal posto da cui si diparte l'arco verso la transazione che scatta. Viceversa, se il peso si riferisce ad un nodo di output, il numero specifica quanti token verranno messi nel posto una volta scattata la transizione.

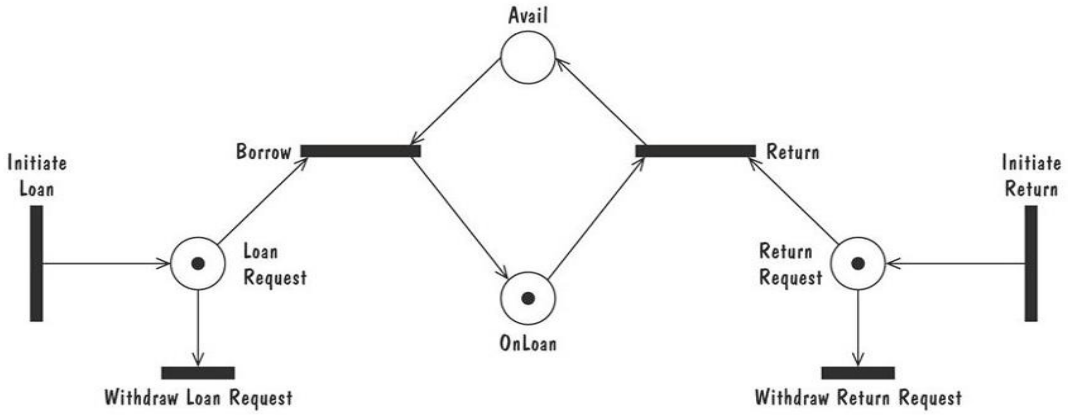
Esempi di configurazioni RdP che rappresentano costrutti di controllo elementari:



Esempio di ciclo:

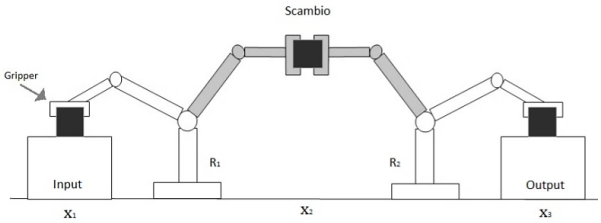


Esempio RdP sistema bibliotecario:

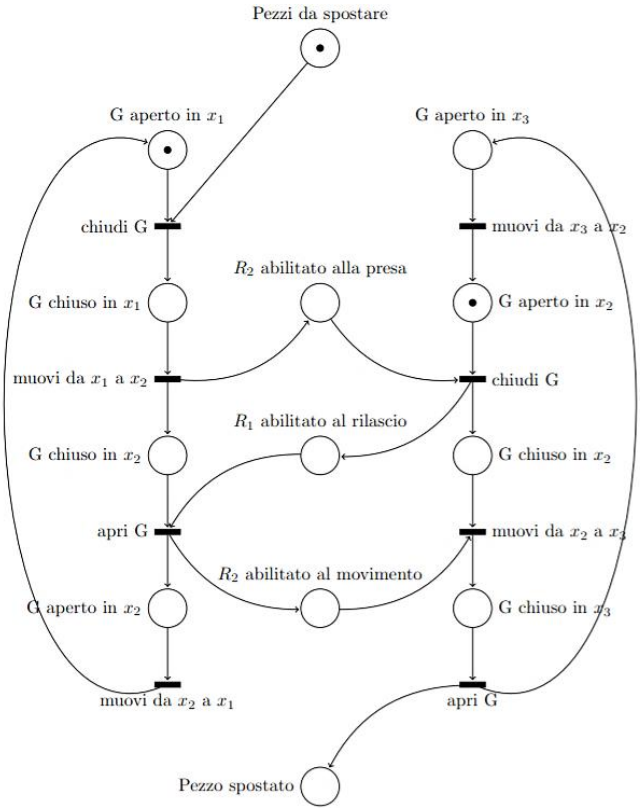


es. Reti di Petri

In figura sono rappresentati due robot, che eseguono in modo coordinato l'operazione PICK-&-PLACE di un oggetto. Ogni robot, per tale operazione, è dotato di una pinza (gripper G) che può essere aperta o chiusa. L'operazione di PICK & PLACE può essere a sua volta scomposta in quattro sotto operazioni:



1. operazione di PICK: R1 effettua il prelievo dell'oggetto dalla stazione di input;
2. trasporto dell'oggetto dalla stazione di input al punto di incontro;
3. scambio dell'oggetto da R1 a R2;
4. trasporto dell'oggetto dal punto di incontro alla stazione di output.



trasformazioni fra modelli possono essere definite e applicate automaticamente da strumenti software.

UML

Modellazione software (software modeling): approcci e motivazioni

- è un approccio pratico, focalizzato sulle tecnologie e strumenti largamente accettati e utilizzati nell'industria.
- permette di costruire modelli indipendenti dal linguaggio (language-indipendent).
- dà vita al paradigma dello sviluppo guidato da modelli (model-driven development).
- ha un grande impatto nella fase di analisi, design e documentazione.

Un modello software (similmente a un modello di sistema) è una semplificazione della realtà, è una descrizione accurata e possibilmente parziale di un sistema in fase di studio ad un qualche livello di astrazione. Un modello software:

- è formato da diversi sotto modelli che descrivono una certa vista del sistema;
- non ha bisogno di essere completo;
- è espresso in qualche linguaggio a qualche livello di astrazione;
- è più di una descrizione: è una rappresentazione analogica di quello che vuole modellare.

I modelli software ci aiutano a visualizzare un sistema così com'è; ci permettono di specificare sia la struttura che il comportamento del sistema. Ci forniscono un template che ci guida durante l'intera costruzione del sistema e inoltre documentano le decisioni prese dal team.

L'**UML** è un **linguaggio di modellazione** standardizzato (ISO) che ci permette di visualizzare, specificare, costruire e documentare artefatti software. UML nasce come strumento di comunicazione nelle fasi di progettazione: un fallimento di comunicazione durante il processo di sviluppo può portare a conseguenze negative. UML è indipendente dalle metodologie di sviluppo.

L'UML non è un linguaggio di programmazione: è indipendente dai dettagli di basso livello (è machine indipendent). UML serve come mezzo di discussione dei problemi (requisiti, analisi, design). UML fornisce diverse viste di diverso livello di dettaglio dello stesso artefatto.

Segue una lista dei principali modelli software utilizzati, in ambito del linguaggio UML

Casi d'uso

Delineo solo i requisiti funzionali che conivolgono INTERAZIONE con l'utente

Un caso d'uso è una descrizione di una serie di sequenze di azioni (comprese le loro varianti), che un sistema esegue per produrre un risultato di valore osservabile per un attore. Un caso d'uso **descrive cosa fa un sistema** ma non specifica come lo fa. Un caso d'uso solitamente rappresenta un pezzo principale di funzionalità che porta valore tangibile all'utente. Un caso d'uso è la descrizione di uno scenario (o di set di scenari strettamente correlati) nel quale il sistema interagisce con l'utente. I casi d'uso sono descritti da scenari narrativi e modelli grafici.

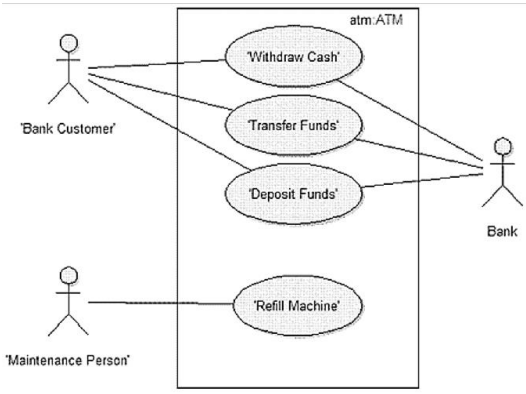
Un **attore** è un agente esterno che deve/vuole interagire con il sistema e che rappresenta uno specifico ruolo. Gli attori NON fanno parte del sistema.

Notazione dei casi d'uso:

- Caso d'uso: ovale
- Attore: stickman

ha senso usarlo solo quando c'è interazione con l'utente. come vedo se è INTERATTIVO? con DFD liv. 0 vedo quali sono gli attori.

Un diagramma dei casi d'uso è un modo eccellente per comunicare alla gestione, ai clienti e alle altre persone che non sviluppano il software cosa farà il sistema quando sarà completato.



I diagrammi dei casi d'uso sono usati per modellare il **contesto** e i **requisiti** di un sistema. Inoltre, forniscono la prospettiva dell'utente del sistema.

Un'**associazione** tra caso d'uso e attore indica che l'attore e il caso d'uso comunicano l'uno con l'altro, possibilmente mandando e ricevendo messaggi. L'associazione descrive il "canale" di interazione. È importante notare che un'associazione:

- non modella un flusso di dati;
- non indica una direzione del flusso di interazione, dunque l'associazione è raramente ornata da una freccia terminale;
- l'interazione è descritta da un testo fuori dal diagramma.

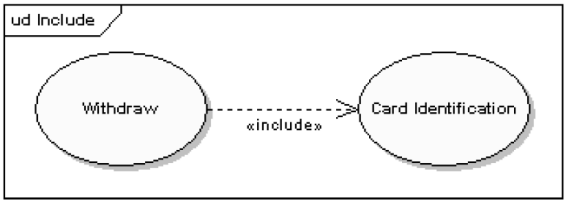
È possibile notare delle associazioni dall'esempio di diagramma di casi d'uso ATM sovrastante.

Relazioni tra casi d'uso

La relazione tra casi d'uso **<<include>>** specifica che il caso d'uso d'origine **incorpora** esplicitamente il comportamento del caso d'uso incluso. Si ha che:

- il caso d'uso d'origine specifica il punto esatto in cui il flusso è sospeso e il caso d'uso è iniettato.
- Alla fine dell'esecuzione del caso d'uso incluso, il caso d'uso d'origine continua la sua esecuzione dal punto in cui è stato sospeso.
- Tutte queste descrizioni sono di natura testuale e quindi posizionate al di fuori del modello grafico.

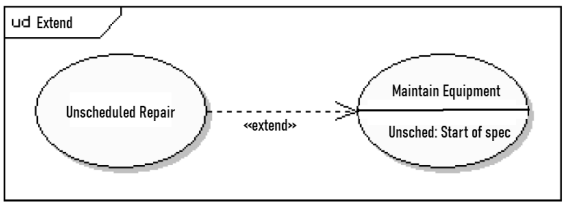
Un esempio è il seguente: "card identification" è incluso in "withdraw"



La relazione tra casi d'uso **<<extend>>** specifica che il caso d'uso di partenza estende il comportamento del caso d'uso di arrivo, aggiungendo una logica personalizzata eccezionale in una posizione specificata dall'origine. Si ha che: NON CENTRA CON EREDITARIETA'

- il punto di estensione specifica dove il caso d'uso d'origine è sospeso; la condizione specifica la regola che avvia l'attivazione del caso d'uso target.
- La riprese dal caso d'uso d'origine è simile al caso di inclusione.

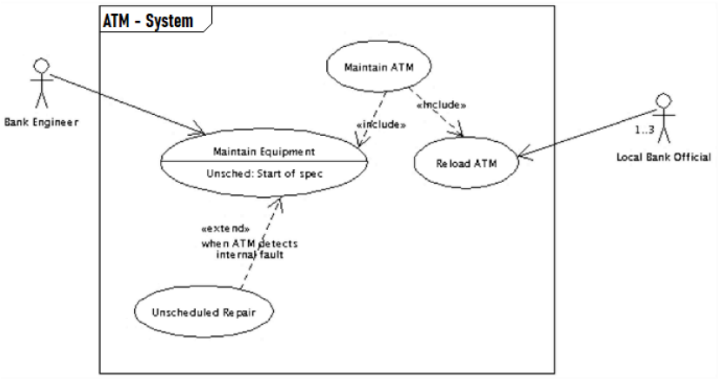
Un esempio è il seguente: "unscheduled repair" estende "maintain equipment"



extension point: es. i motivi per cui va in failure

Notazione: si noti che le frecce di <<include>> ed <<extend>> sono tratteggiate e aperte all'estremità.

Esempio complessivo di caso d'uso:

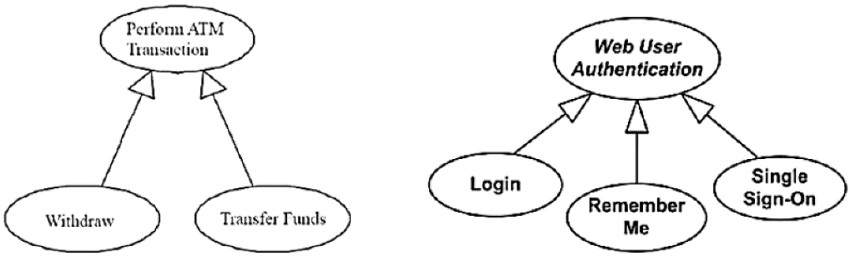


Sempre ritornando alla relazione extend, in relazione all'esempio:

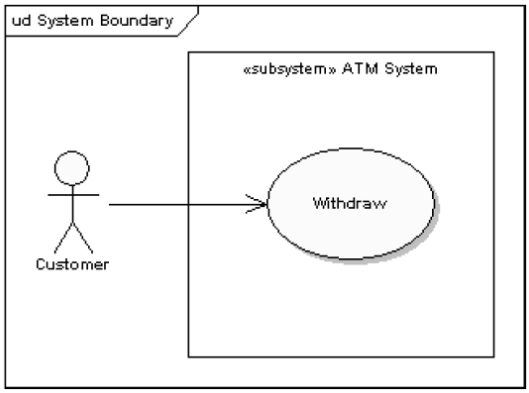
- "when ATM detects internal fault" è la **condition** per la relazione extend;
- "unsched: start of spec" è l'**extension point** per la relazione extend;

La relazione tra casi d'uso di **generalizzazione** specifica che un caso d'uso (figlio) eredita la struttura, il comportamento e le relazioni di un altro attore padre. Il figlio è il caso d'uso più specializzato, mentre il padre è quello più astratto della relazione. La notazione è una freccia completa con l'estremità chiusa dal caso figlio al caso padre

Esempi di generalizzazione:



I **confini del sistema** sono usati per definire confini concettuali, aiutano a raggruppare elementi correlati logicamente e aiutano a mostrare cosa risiede dentro il sistema (le funzionalità) e cosa sta fuori (l'utente).



Se un caso d'uso e' troppo complesso (non riesco a visualizzarlo tutto in una schermata o e' troppo complesso) lo posso partizionare in piu' livelli

Guidelines

Segue una serie di linee guida per i diagrammi dei casi d'uso:

- Iniziare il nome dei casi d'uso con un verbo (affermare il valore reale per l'attore)
- Cercare di evitare verbi vaghi come gestire, processare ed eseguire
- Sfruttare il vocabolario del dominio
- Fare attenzione a un usare troppi livelli nidificati di casi d'uso di inclusione, questo significa focalizzarsi sui dettagli di implementazione (come) che non è lo scopo dei casi d'uso (cosa).
    - in linea generale: max. un livello nidificato.

Diagramma delle classi

Una **classe** è un template per la definizione di tutte le caratteristiche di un oggetto, come attributi e metodi. Una classe definisce una serie di oggetti con la stessa struttura comune e lo stesso comportamento.

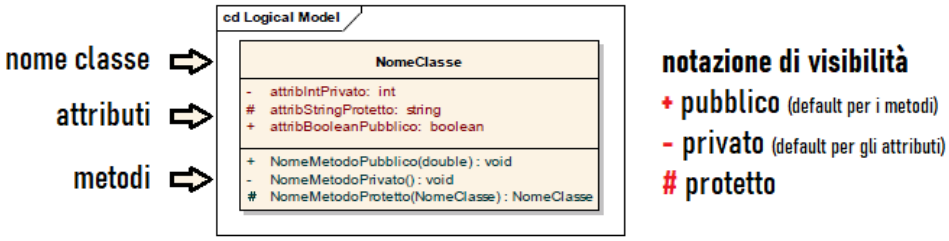
Un **attributo** descrive un'informazione relativa allo stato interno di un oggetto (istanza della classe). Ogni oggetto mantiene un valore per ciascun attributo per la corrispondente classe di appartenenza.

Un **metodo** definisce il comportamento di un oggetto per uno specifico evento. Il comportamento di un oggetto può essere indipendente dallo stato o dipendente dallo stato.

Un **diagramma delle classi** descrive le tipologie di oggetti presenti in un sistema e le relazioni statiche che esistono tra di loro. È una tecnica di modellazione basata sui principi dell'orientazione agli oggetti. È la più ricca notazione in UML.

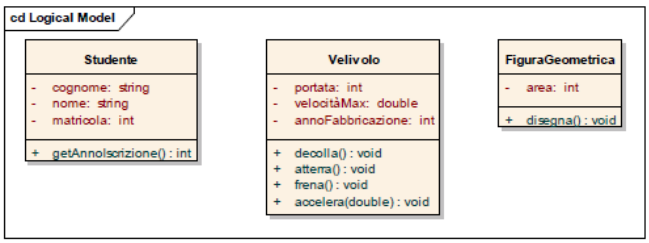
Segue la notazione grafica UML del diagramma delle classi:

Una **classe** è una descrizione di un insieme di oggetti che hanno attributi, operazioni, relazioni e comportamenti simili. Tipicamente una classe descrive un'entità del dominio del problema.



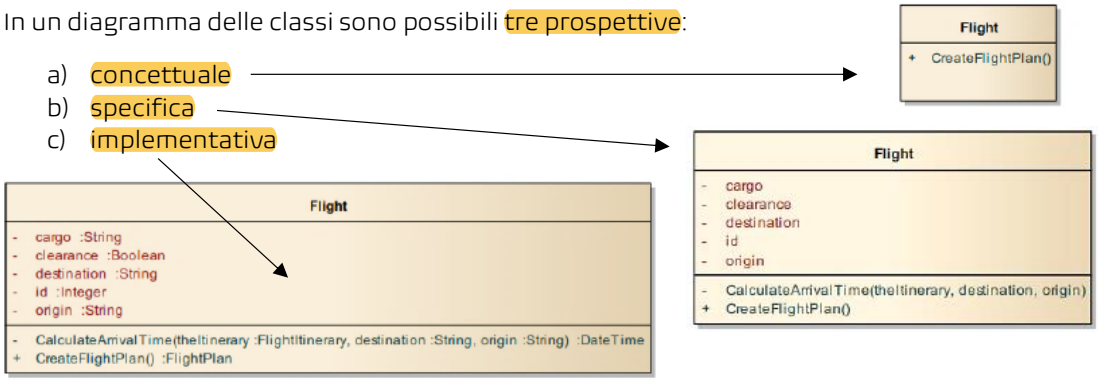
convenzioni (linee guida) sui nomi:

- Classe: iniziale maiuscola per ogni parola
- Attributi: iniziale minuscola, maiuscola per ogni singola parola successiva
- Metodi:
    - C++: maiuscola anche la prima
    - Java: come gli attributi



In un diagramma delle classi sono possibili tre prospettive:

a) concettuale
b) specifica
c) implementativa



La relazione di **associazione** è una relazione "semantica" tra due o più classi che specifica una connessione tra le loro istanze. È una relazione strutturale che specifica che due oggetti di una classe sono connessi a oggetti di un'altra classe (possibilmente anche la stessa).

Esistono relazioni mono o bi-direzionali:

- **origine** e **target** (classe di origine e destinazione)
- gli end-point possono includere delle etichette con numeri o descrizioni.

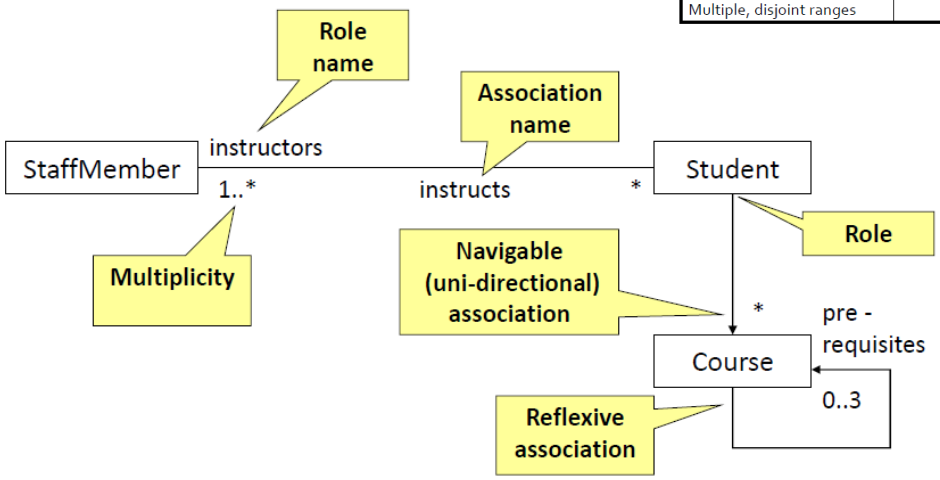
Un'associazione tra due classi specifica che l'oggetto di un end-point conosce l'oggetto dell'altro end-point, e sono capaci di scambiarsi messaggi. Notazioni opzionali di un'associazione:

- **nome dell'associazione** (opzionale): etichetta posizionata a metà della relazione che rappresenta un verbo che fornisce l'implicita azione della relazione.
- **nome del ruolo** (opzionale): etichetta posizionata alla fine degli end-point che specifica il ruolo dell'end-point nel contesto dell'associazione.

Molteplicità: indica quanti oggetti di una classe possono far riferimento ad ogni oggetto dell'altra. Permette di indicare se un'associazione è obbligatoria o meno, e il lower e upper bound del numero di istanze.

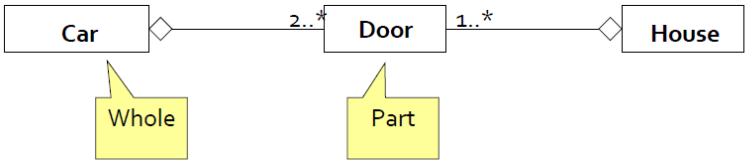
Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

Esempio:



Aggregazione: è una forma speciale di associazione che modella una relazione "whole-part" tra un aggregato (whole – il tutto) e le sue parti. Modella il concetto di "è parte di" e "contiene".

Esempio: car-door; house-door;

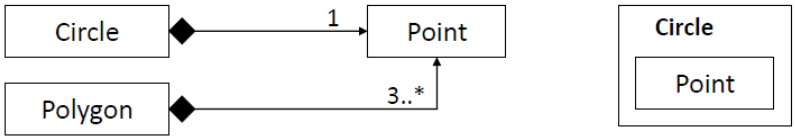


Composizione: è una forte forma di contenimento (più forte dell'aggregazione).

- il "whole" è l'unico proprietario di ogni parte.
- le molteplicità nel "whole" devono essere 1 o 0.

Mentre in AGGREGAZIONE posso avere ISTANZE CONDIVISE con COMPOSIZIONE le ISTANZE sono DISTINTE E SERIALI

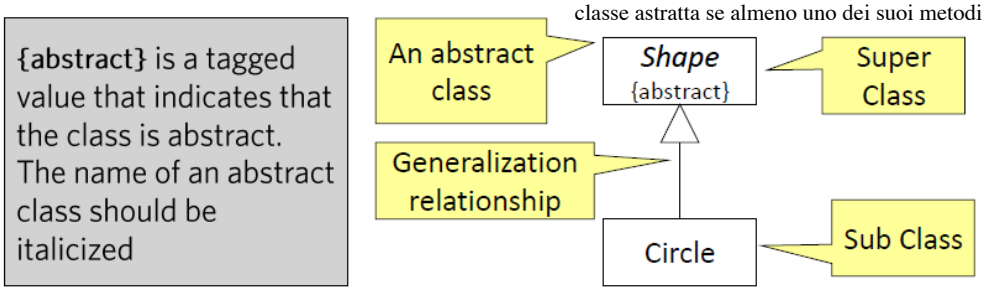
Esempio:



Generalizzazione: indica che oggetti di una classe specializzata (**sottoclasse – subclass**) sono sostituibili da oggetti di una classe generale (**classe generale - superclass**).

Una sottoclasse eredita dalla superclasse: attributi, operazioni e relazioni.
Una sottoclasse può aggiungere operazioni e relazioni, rifinire (sostituire – override) le operazioni ereditate dalla superclasse.

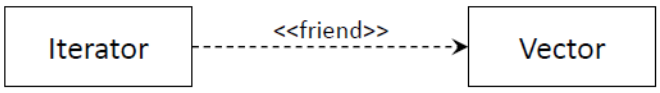
classe astratta se almeno uno dei suoi metodi e' astratto (non implementato)

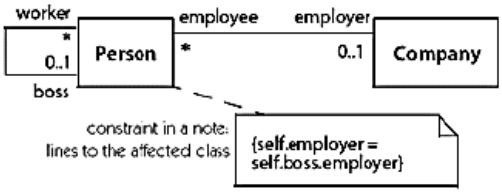


Una relazione di **realizzazione** indica che una classe implementa un comportamento specifico di qualche interfaccia. Un'interfaccia può essere realizzata da più classi. Una classe può realizzare più interfacce. Con interfaccia ci si riferisce al concetto di "insieme di comportamenti visibili".

Un figlio puo' estendere solo un padre

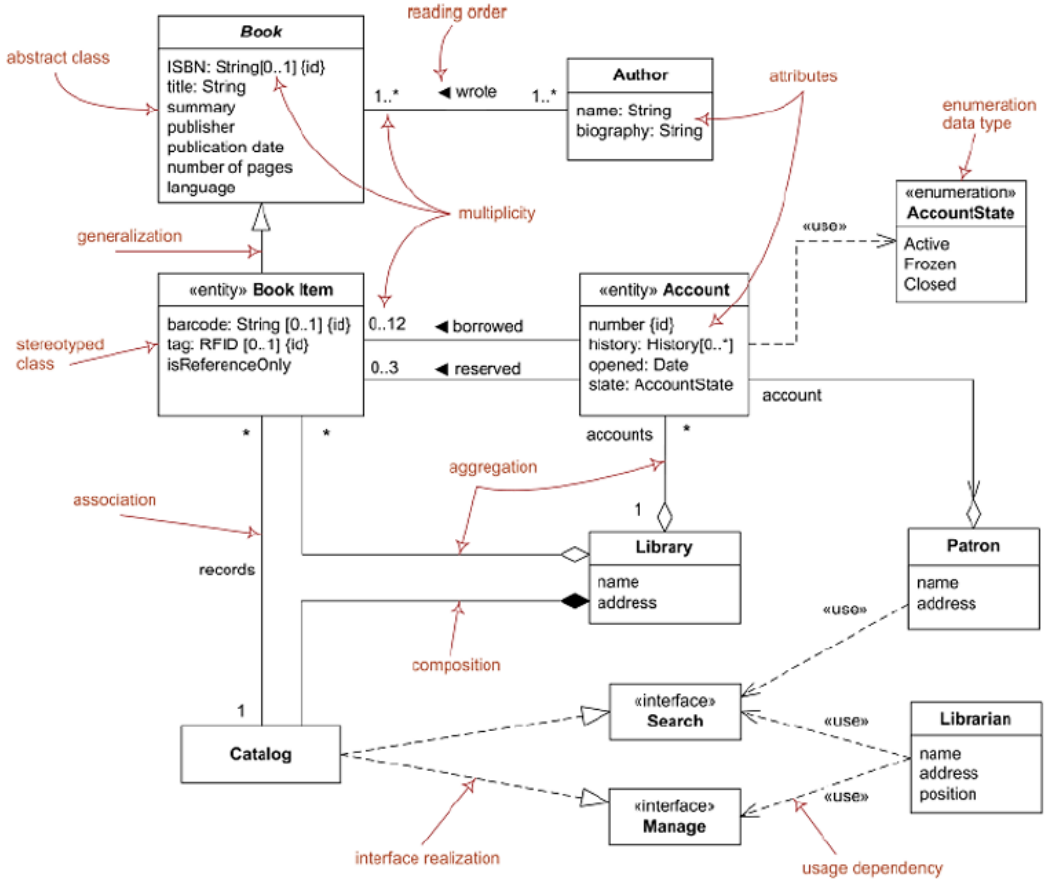
Una **dipendenza** indica una relazione tra due classi sebbene non ci sia un'esplicita associazione tra di loro. È una forma più debole di accoppiamento.



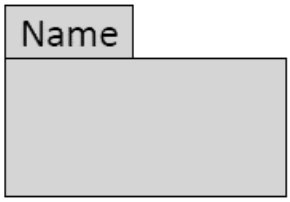


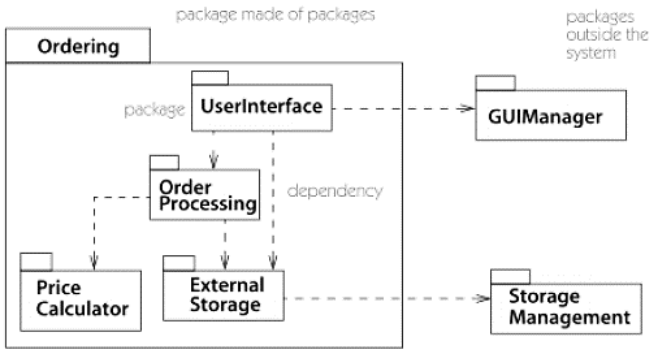
I **vincoli** (**constraints**) sono note che pongono restrizioni semantiche annotate come boolean expression. I vincoli sono utilizzati per rappresentare assunzioni, descrivere invarianti, descrivere pre e post condizioni dei metodi.

Esempio riassuntivo della notazione:



Un **package** è un meccanismo di raggruppamento general purpose. Viene tipicamente utilizzato per specificare l'architettura logica del sistema. Un package non si traduce per forza in un sottosistema fisico.

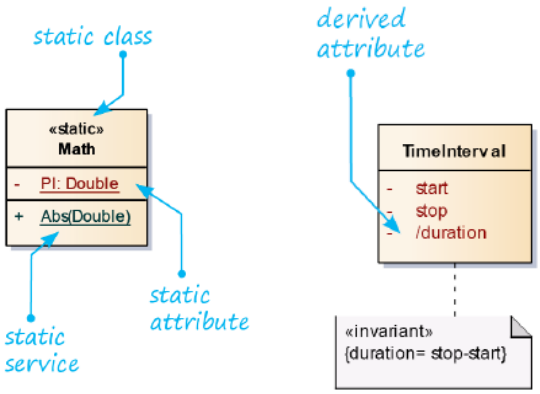




Static, derived, red only, frozen:

Vincoli:

- **{readOnly}** è un vincolo che precisa che il valore dell'attributo a cui è riferito non può essere modificato dall'esterno.
- **{frozen}** è un vincolo che precisa che il valore dell'attributo a cui è riferito non può cambiare per tutta la vita di un oggetto. (costante)



Diagrammi di sequenza

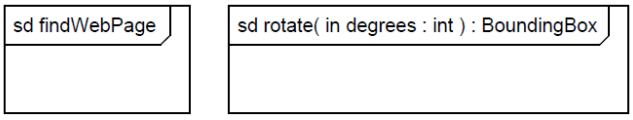
Un diagramma di sequenza descrive le interazioni tra gli oggetti coinvolti in uno specifico scenario. Gli oggetti collaborano per portare a termine un task o un caso d'uso. Gli oggetti collaborano scambiandosi messaggi. L'intero insieme di oggetti e le loro mutue interazioni in uno specifico scenario viene chiamato **collaborazione**. Il meccanismo di scambio messaggi nel paradigma OO è analogo a quello delle chiamate a funzioni nel paradigma procedurale.

I diagrammi di sequenza sono utili per modellare:

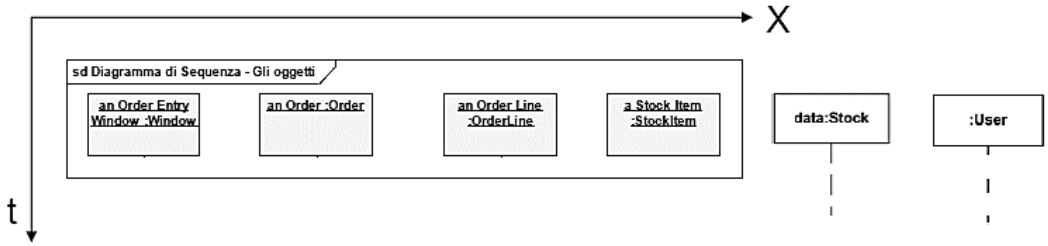
- interazioni in fase di progettazione "mid-level";
- le interazioni tra un prodotto e il suo ambiente;
- le interazioni tra componenti di un sistema nella progettazione architetturale.

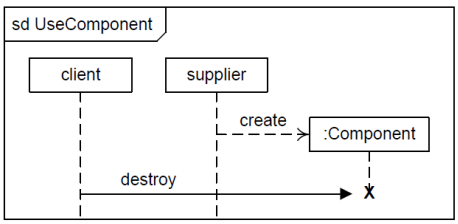
Segue la notazione:

Un **frame** è un rettangolo con un pentagono nella parte superiore sinistra che indica il nome del compartimento (contesto).



Gli individui che partecipano sono disposti nel diagramma sotto forma di linee temporali. La linea verticale indica il tempo; la linea tratteggiata indica il periodo di tempo in cui un certo individuo del diagramma esiste.





- un **nuovo oggetto** compare nel punto in cui viene creato.
- un **oggetto distrutto** riceve un troncamento nella sua linea temporale con una X all'estremità.
- un **oggetto "che vive"** ha una linea tratteggiata lungo tutta la linea temporale.

Frecce:

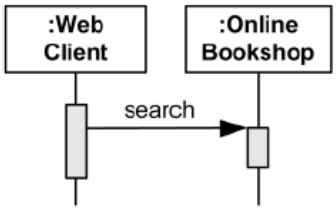
- Sincrono: il mittente sospende l'esecuzione finché il messaggio non è completato.

- Asincrono: il mittente continua l'esecuzione dopo aver inviato il messaggio.

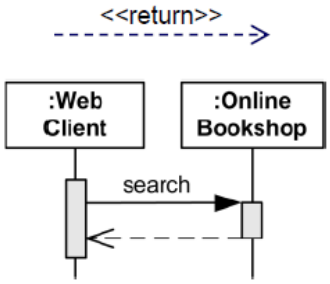
- Il messaggio sincrono ritorna o viene creata un'istanza.

Chiamate sincrone:
Frecce dall'estremità chiusa dal chiamante al chiamato. La freccia è adornata da una tabella con il nome del servizio chiamato (opzionalmente è possibile mostrare argomenti e valori di ritorno). Il chiamante aspetta la terminazione del servizio chiamato prima di continuare (sincrono). Il **tempo di attivazione** è mostrato da un rettangolo sopra la corrispettiva linea temporale dell'oggetto.

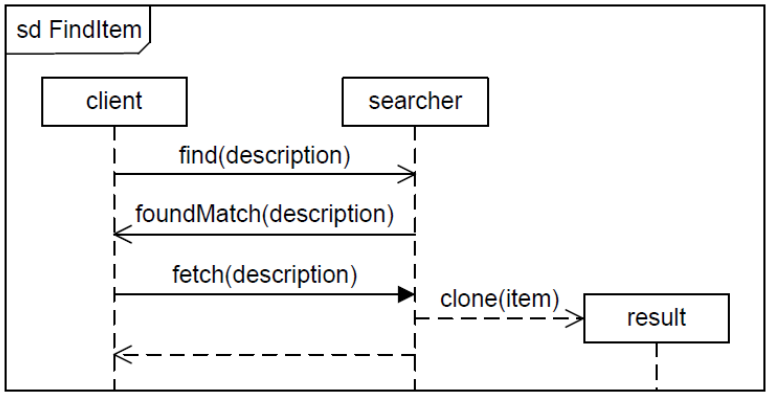
l'oggetto WebClient richiama il metodo search dell'oggetto OnlineBookshop



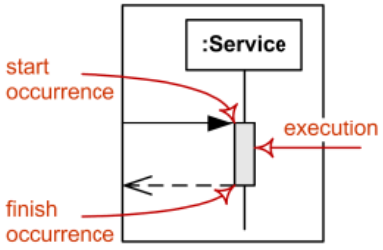
Messaggio di ritorno:
Il messaggio di ritorno è mostrato con una freccia tratteggiata dall'estremità aperta. Il messaggio di ritorno è opzionale, se non mostrato, la terminazione dell'esecuzione è determinata dalla fine del box di attivazione.



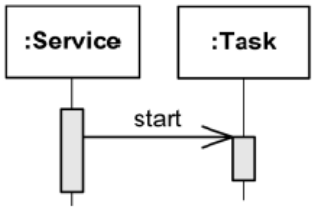
Esempio:



Istanze di esecuzione:
Una operazione è in esecuzione quando qualche processo sta eseguendo il proprio codice. Una operazione è sospesa quando invia un messaggio sincrono ed è in attesa del return. Una operazione è attiva quando sta eseguendo o è sospesa. Il periodo in cui un oggetto è attivo può essere mostrato usando un'istanza di esecuzione: un rettangolo sopra la linea temporale:

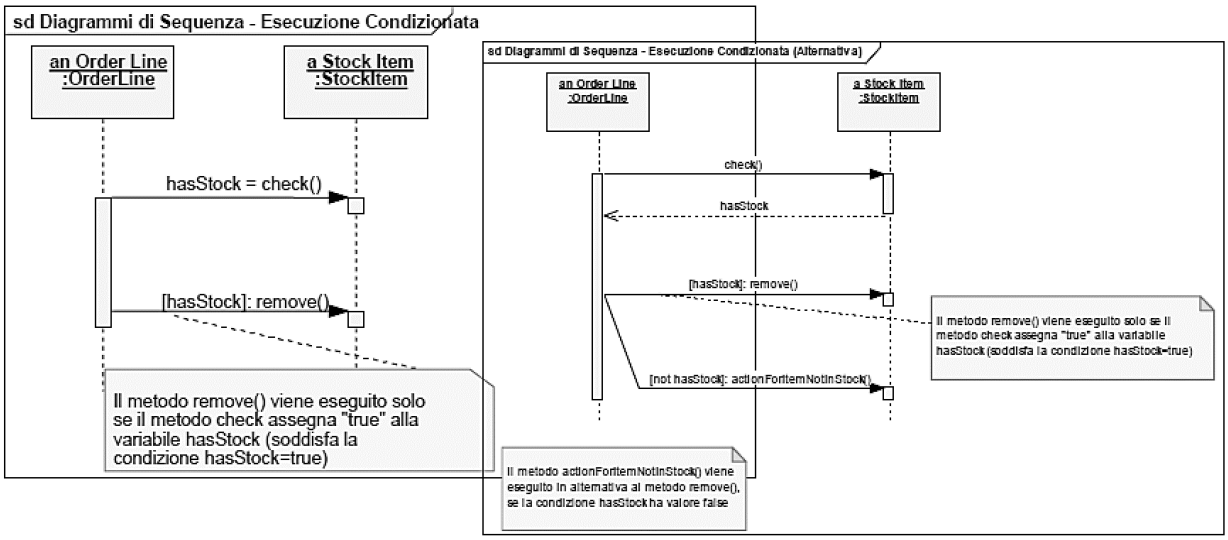


Chiamate asincrone:
Adatte per interazioni concorrenti. La notazione è una freccia con l'estremità aperta dal chiamante al chiamato. La freccia è adornata dal nome del servizio invocato, e opzionalmente gli argomenti di input e i valori ritornati. Il chiamante può procedere l'esecuzione dopo aver inviato il messaggio.



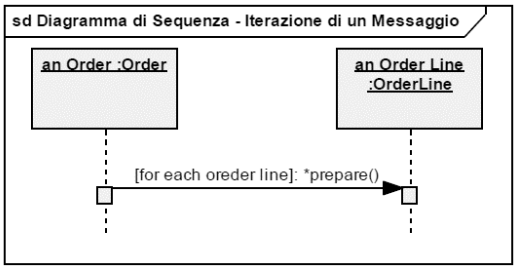
Chiamate condizionali:
L'invio di un messaggio può essere soggetto a una condizione che viene controllata a run-time.

Sintassi: **(condizione) : nomeMetodo()**

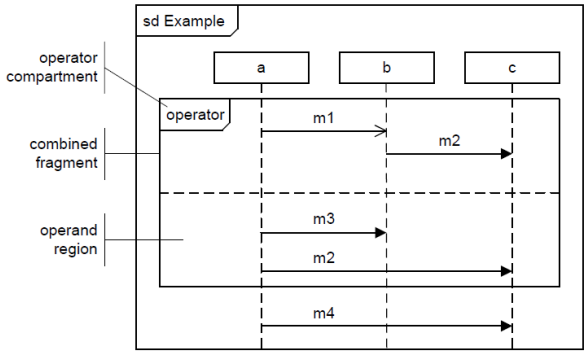


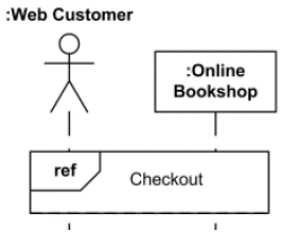
Messaggio di interazione:
Esecuzione ciclica di un solo messaggio; la "guardia" specifica quando il loop deve terminare.

Sintassi: **(condizione) : * nomeMetodo()**



Frammenti combinati (combined fragments):
È una parte marcata di una interazione che mostra: rami, loop, esecuzioni concorrenti etc.



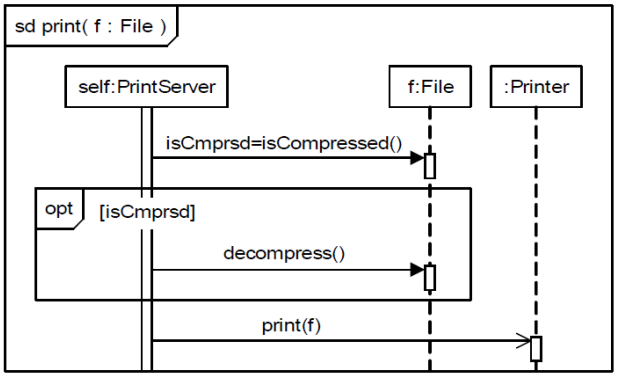


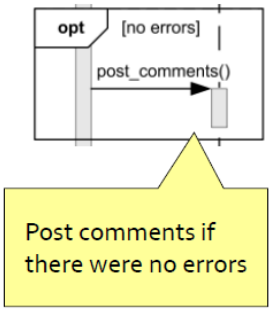
L'**uso di interazione** è un frammento che permette di usare (o chiamare) un'altra interazione. L'uso di interazione è un frammento combinato con l'operatore: **ref**.

I **frammenti opzionali** sono una porzione di interazione che potrebbe non essere eseguita; sono equivalenti alle dichiarazioni condizionali. L'operatore è: **opt.** con un solo operando con una guardia.

Una **guardia** è un un'espressione booleana racchiusa tra parentesi quadre in un formato non specificato da UML.

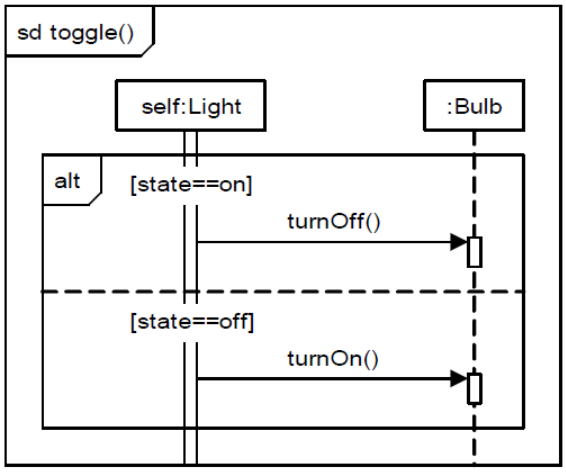
Esempio: IF

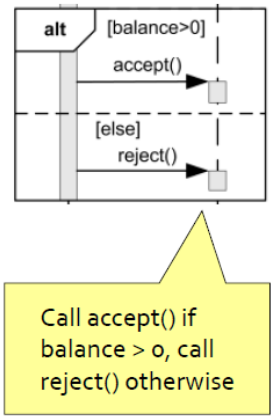




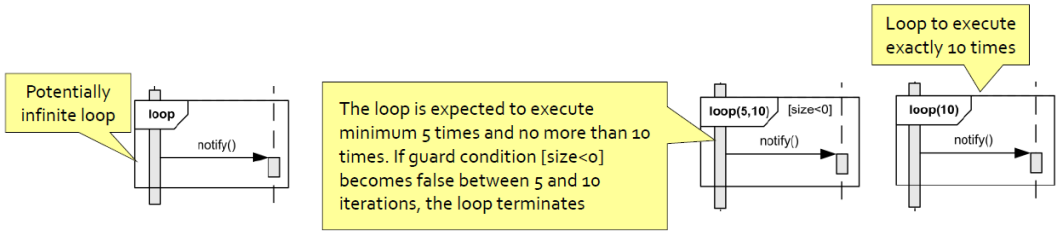
I **frammenti alternativi** con una o più guardie come operandi sono mutuamente esclusivi. L'operatore è: **alt**.

Esempio: SWITCH



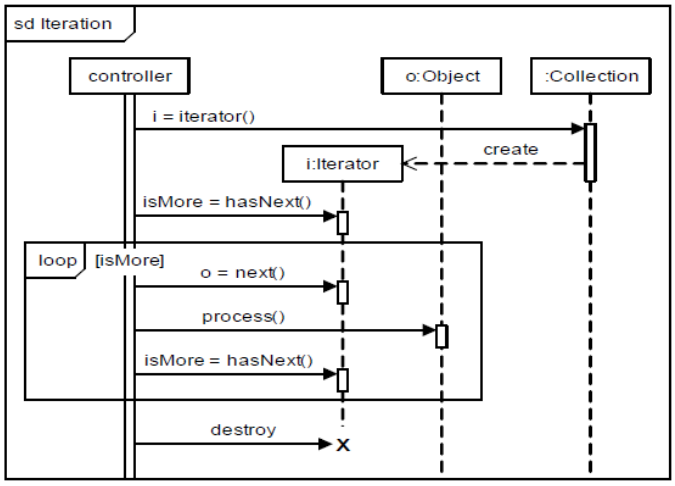


I **frammenti di loop** sono corpi singoli che possono avere un operando guardia. L'operatore ha la seguente forma: **loop (min, max)** dove i parametri sono opzionali.



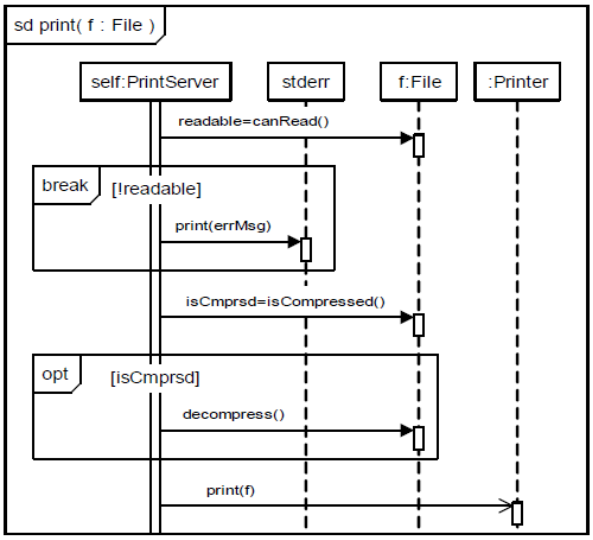
Il corpo del loop viene eseguito almeno min volte e al massimo max volte. Se il corpo del body è stato eseguito almeno min volte ma meno di max volte, viene eseguito solo se la guardia è true.

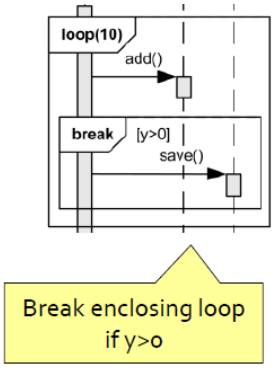
Esempio:

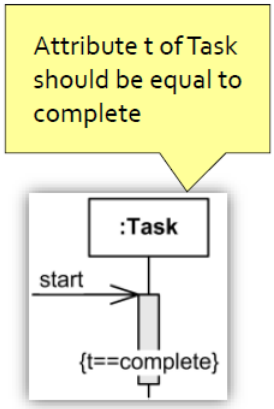


I **frammenti di rottura** (**break fragment**) sono frammenti combinati in cui un operando ha eseguito al posto di un operando o diagramma racchiuso se la guardia è vera. L'operatore è: **break**.

Esempio:







Le **invarianti di stato** sono un frammento di interazione che rappresentano vincoli run-time per il partecipante dell'interazione. Possono essere usati per specificare differenti tipologie di vincoli, come valori di attributi/variabili o stati interni/esterni etc. Il vincolo è valutato immediatamente precedentemente all'esecuzione della prossima occorrenza specificata così che tutte le azioni non esplicitamente modellate sono state eseguite.

Esempio di un diagramma di sequenza completo: