

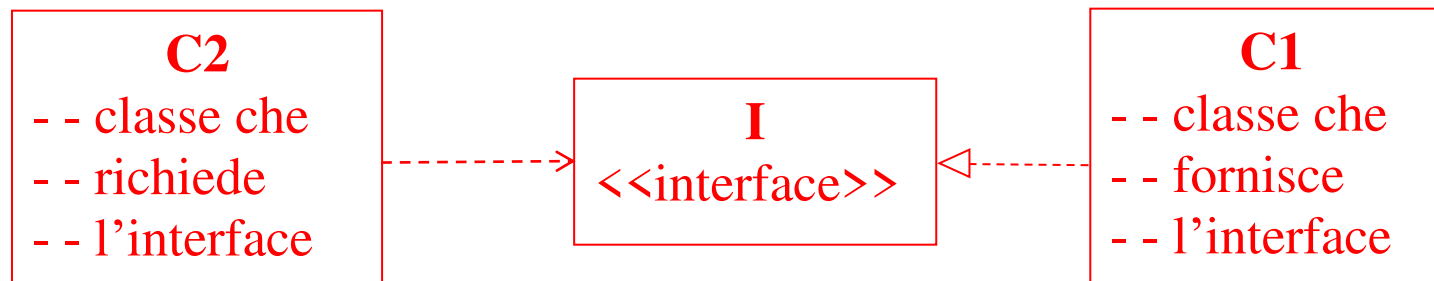
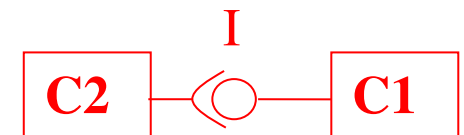
Rapporti fra classi e interfacce

Una classe:

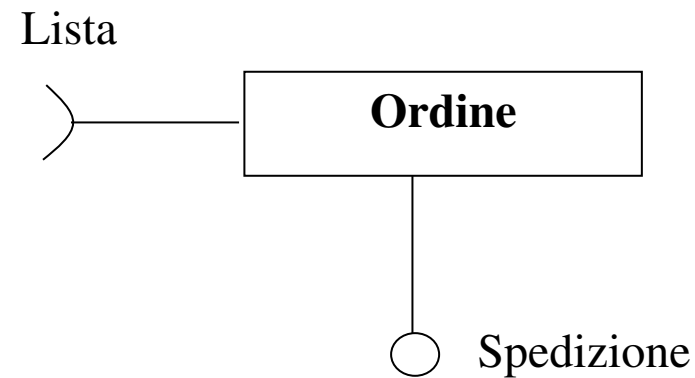
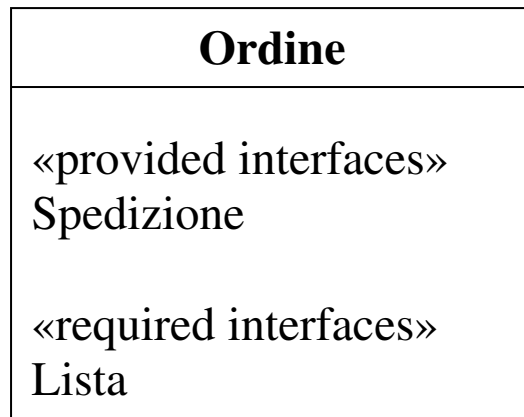
- *fornisce una interfaccia* se implementa tale interfaccia;
- *richiede una interfaccia* se dipende da essa (cioè ha bisogno di una sua istanza per funzionare).

In UML \exists due tecniche **aggiuntive** per evidenziare le interfacce fornite e richieste da una singola classe:

- Elencare tali interfacce entro il riquadro della classe
- Usare la notazione “socket e pallina”



Rappresentazioni alternative



Diagrammi delle strutture composite

- Novità di UML 2
- Consentono di spezzare classi/componenti complessi in parti più piccole, evidenziando le interfacce fornite e quelle richieste da ciascuna parte
- Spesso usati nei diagrammi dei componenti

Struttura composita (fa riferimento al raggruppamento di oggetti durante l'esecuzione) \neq package (raggruppamento di classi al momento della compilazione)

Diagramma delle strutture composite (cont.)

Elementi	Sintassi	Semantica
Parte	<p>Rettangolo</p> <ul style="list-style-type: none">• contenuto entro il rettangolo della composizione che si sta descrivendo• contenente <i>nome : classe</i>, dove sia <i>nome</i>, sia <i>: classe</i> sono opzionali ma non possono mancare entrambi e sono scritti in grassetto• dotato opzionalmente di molteplicità indicata nell'angolo in alto a destra	<p>Elemento di una composizione (il tutto), dove quest'ultima è una classe o un componente</p> <p>La molteplicità indica il numero di istanze della parte</p>

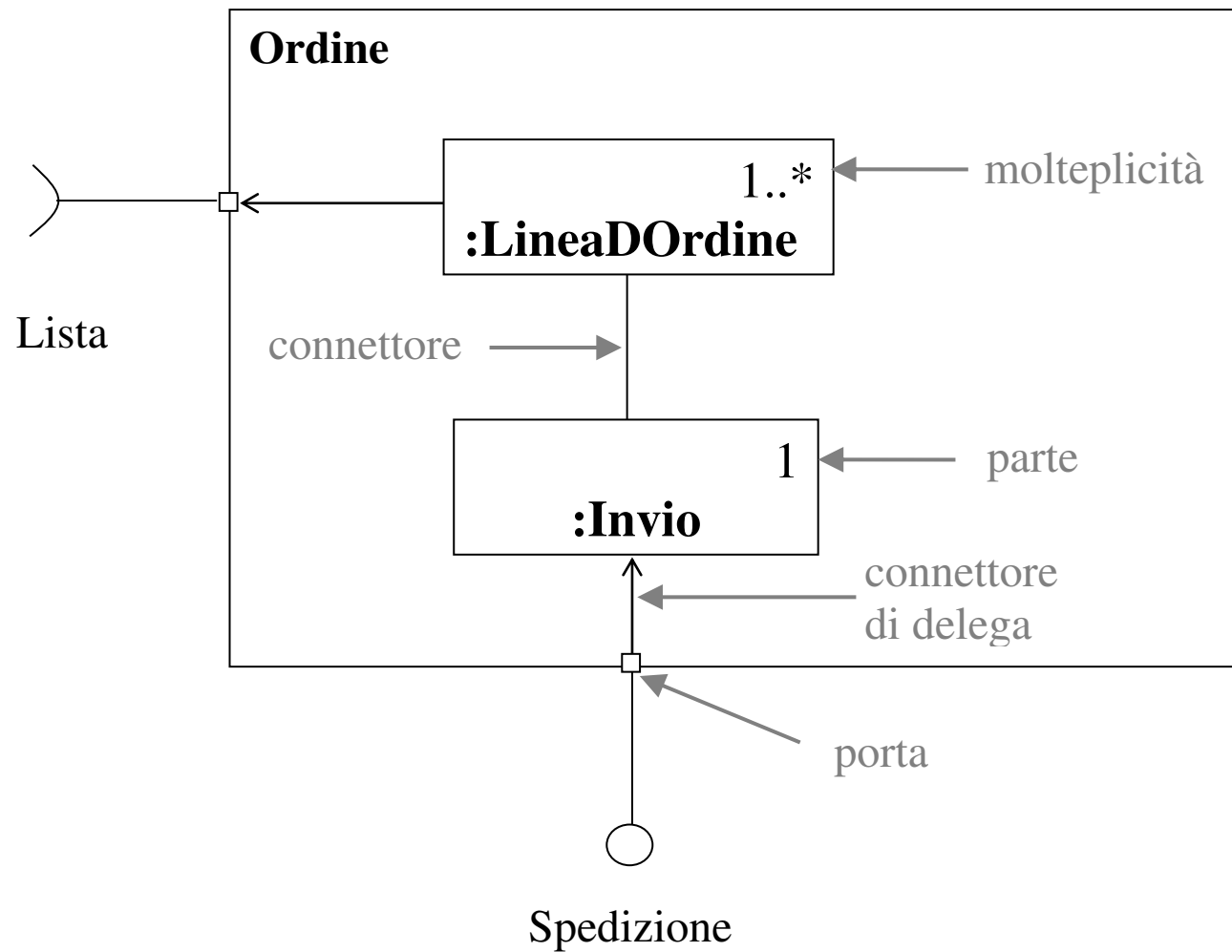
Diagramma delle strutture composite (cont.)

Elementi	Sintassi	Semantica
Porta	Quadrato sovrapposto al bordo del riquadro della classe/componente, collegato a una o più interfacce, dove il collegamento a ciascuna avviene mediante una linea continua distinta e termina o su un socket o su una pallina; dotato opzionalmente di <i>nome</i> (stringa di caratteri)	<p>Punto di comunicazione fra la composizione e il mondo esterno</p> <p>Il collegamento di una porta a più interfacce si usa quando nella rappresentazione di un componente non sono evidenziate le parti interne dello stesso</p> <p>Il nome esplicita l'interazione supportata dalla porta (ad es. visualizzazione)</p>
Connettore	Linea continua (eventualmente dotata di notazione “socket e pallina”) che connette fra loro due parti	<p>Un connettore con “socket e pallina” è utile soprattutto nei diagrammi dei componenti</p>

Diagramma delle strutture composite (cont.)

Elementi	Sintassi	Semantica
Connettore di delega	<p>Freccia con linea continua e punta biforcuta che connette una parte con una porta:</p> <ul style="list-style-type: none">• entra nella porta se alla porta è connessa una interfaccia richiesta (o più interfacce, tutte richieste)• esce dalla porta se alla porta è connessa una interfaccia fornita (o più interfacce, tutte fornite)	Evidenzia quali parti richiedono/forniscono quali interfacce

Esempio



Capitolo 2

Dall'idea al codice con UML 2

Esercizi introduttivi

Obiettivo

Identificare le classi (solo quelle della logica di dominio) per risolvere il problema proposto

1. Valutazione di polinomi

Un polinomio, identificato da una lettera minuscola dell'alfabeto, è dato dalla somma di uno o più monomi.

Un monomio è caratterizzato da un segno, un coefficiente e un grado.

Il programma deve leggere il nome del polinomio p e il valore della variabile per il quale si vuole calcolare il valore di p .

Valutazione di polinomi

Ad esempio, dato il polinomio

$$f: 3x^4 - 5x^3 + 2x + 1,$$

l'utente deve inserire da tastiera

$f(2)$

e il programma deve stampare 13,

poiché

$$f(2) = 3 * 2^4 - 5 * 2^3 + 2 * 2 + 1.$$

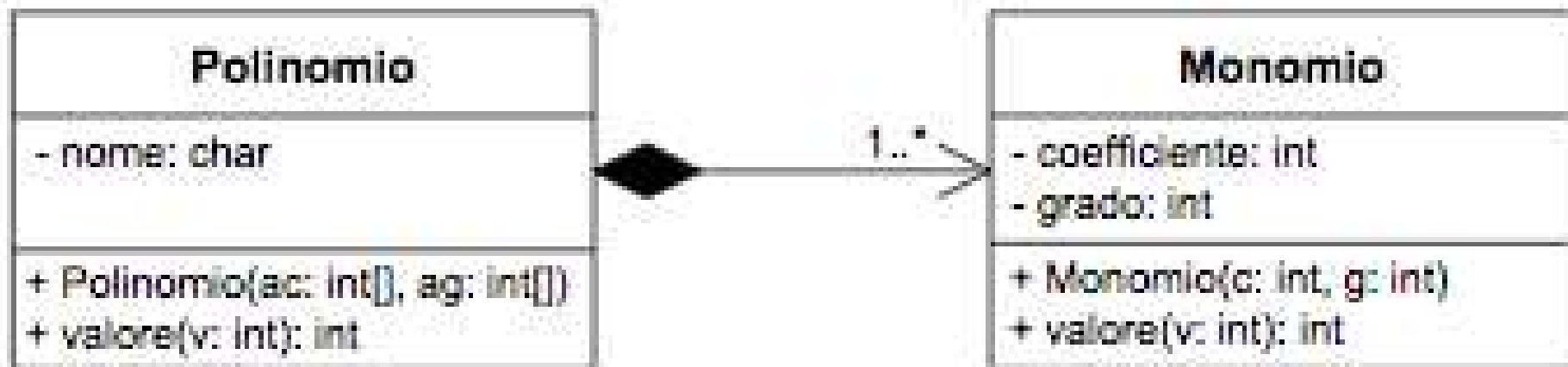
1. Valutazione di polinomi

Un **polinomio**, identificato da una **lettera** minuscola dell'alfabeto, è dato dalla somma di uno o più monomi.

Un **monomio** è caratterizzato da un **segno**, un **coefficiente** e un **grado**.

Il programma deve leggere il nome del polinomio p e il valore della variabile per il quale si vuole **calcolare il valore** di p .

DELEGA

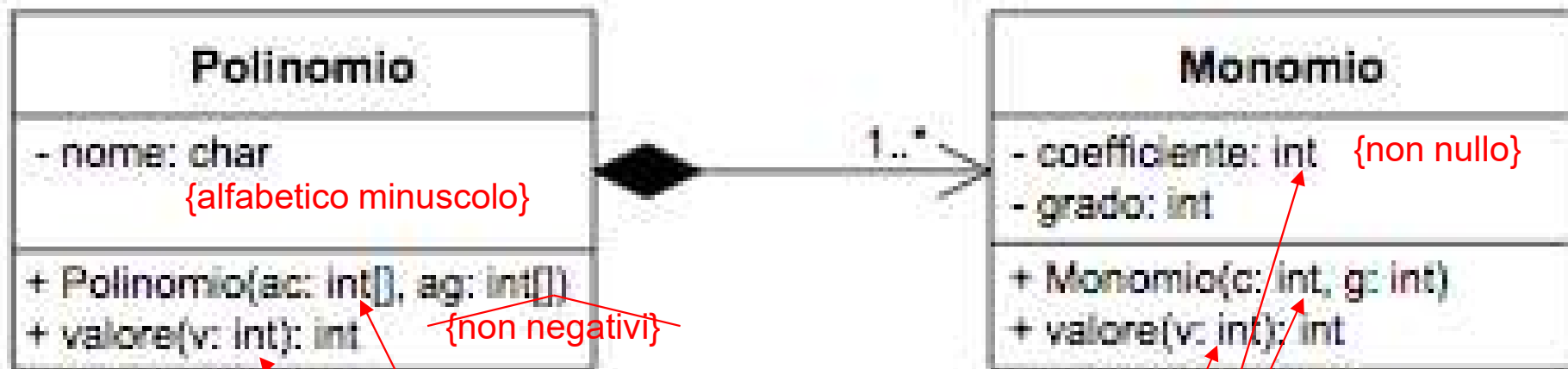


Esercizio

Scrivere il codice Java relativo alle due classi illustrate

- $f: 3x^4 - 5x^3 + 2x + 1$
- $ac [3, -5, 2, 1]$
- $ag [4, 3, 1, 0]$

{numero elementi di ac = numero elementi di ag }



Manca qualcosa?

Forse manca un parametro del costruttore Polinomio, quello relativo al nome (che è un singolo carattere), a meno che tale nome non sia scelto dall'applicazione

Questioni ignorate

- Salvataggio dei polinomi
- Numero massimo dei polinomi
(= numero lettere minuscole?)
- Numero massimo di monomi per polinomio
- Numero di monomi del polinomio corrente (attributo?)

2. CAD tridimensionale

L'applicazione permette di gestire un modello tridimensionale composto da una serie di oggetti.

Ogni oggetto è caratterizzato da una posizione tridimensionale, da un colore e da un materiale.

Esistono tre oggetti di base: sfere, parallelepipedi e tetraedri.

CAD tridimensionale

È altresì possibile avere oggetti complessi composti da un insieme di oggetti (a loro volta complessi o di base).

Ad esempio, due sfere potrebbero creare un rudimentale “pupazzo di neve”.

CAD tridimensionale (cont.)

Deve essere possibile la creazione di oggetti complessi, l'aggiunta di nuovi elementi a un oggetto complesso, l'aggiunta di nuovi oggetti al modello tridimensionale, lo spostamento di oggetti e la loro rotazione rispetto a un asse (x, y, o z).

2. CAD tridimensionale

L'applicazione permette di gestire un **modello tridimensionale** composto da una serie di oggetti.

Ogni **oggetto** è caratterizzato da una **posizione** tridimensionale, da un **colore** e da un **materiale**.

Esistono tre oggetti di base: **sfere**, **parallelepipedi** e **tetraedri**.

CAD tridimensionale

È altresì possibile avere **oggetti complessi** composti da un insieme di oggetti (a loro volta complessi o di base).

Ad esempio, due sfere potrebbero creare un rudimentale “pupazzo di neve”.

CAD tridimensionale (cont.)

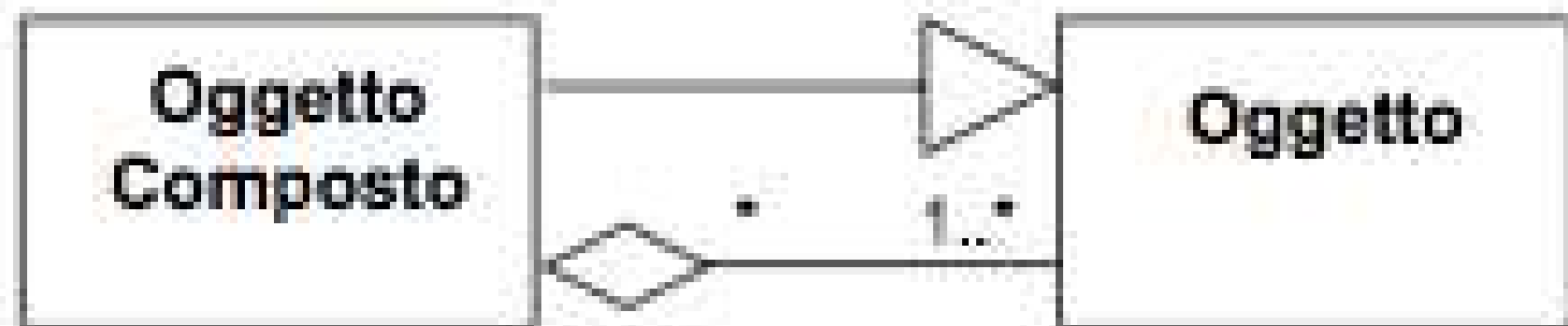
Deve essere possibile la creazione di oggetti complessi, l'aggiunta di nuovi elementi a un oggetto complesso, l'aggiunta di nuovi oggetti al modello tridimensionale, lo spostamento di oggetti e la loro rotazione rispetto a un asse (x, y, o z).

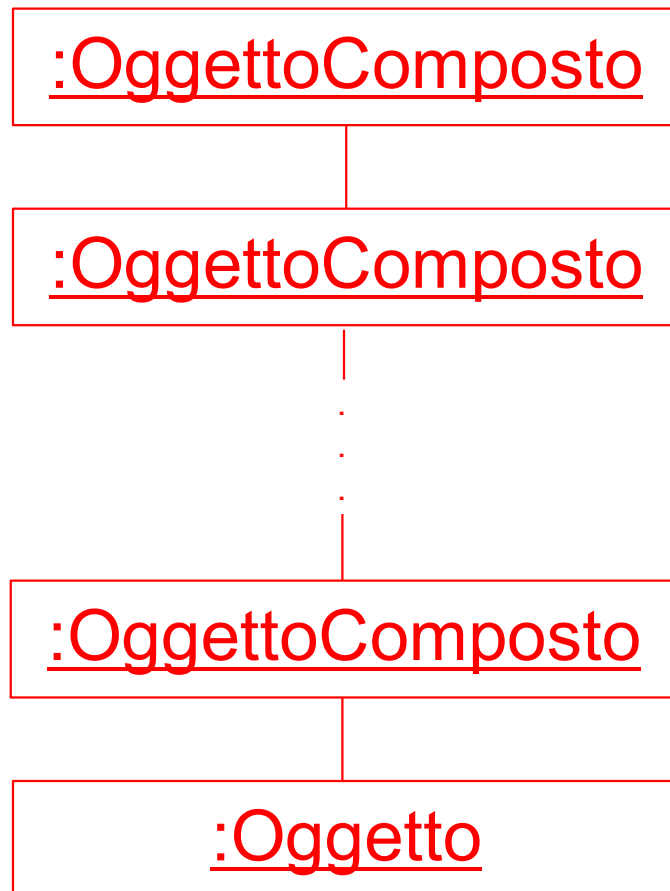
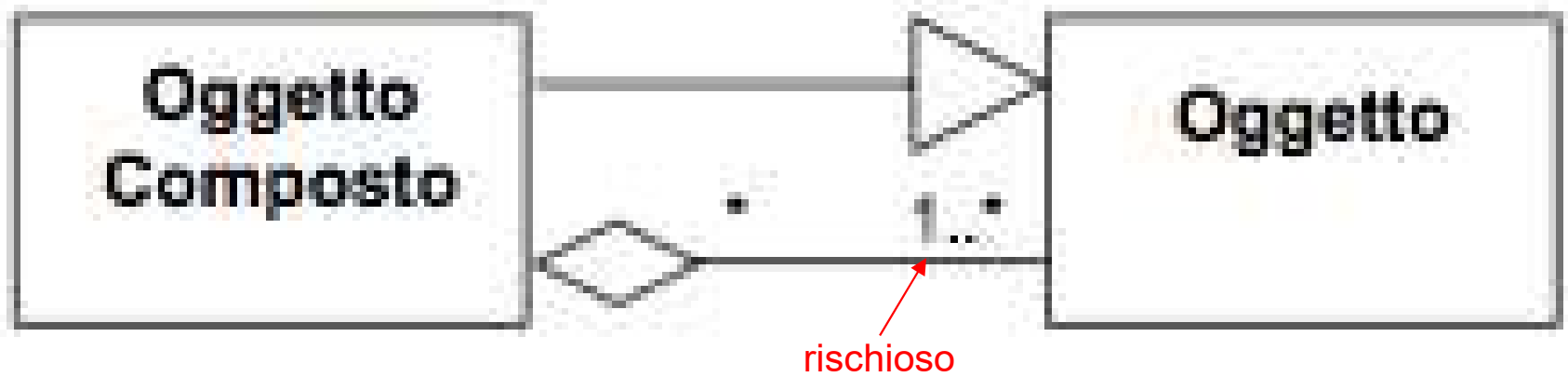
Vincoli aggiunti esplicitamente dal libro di testo

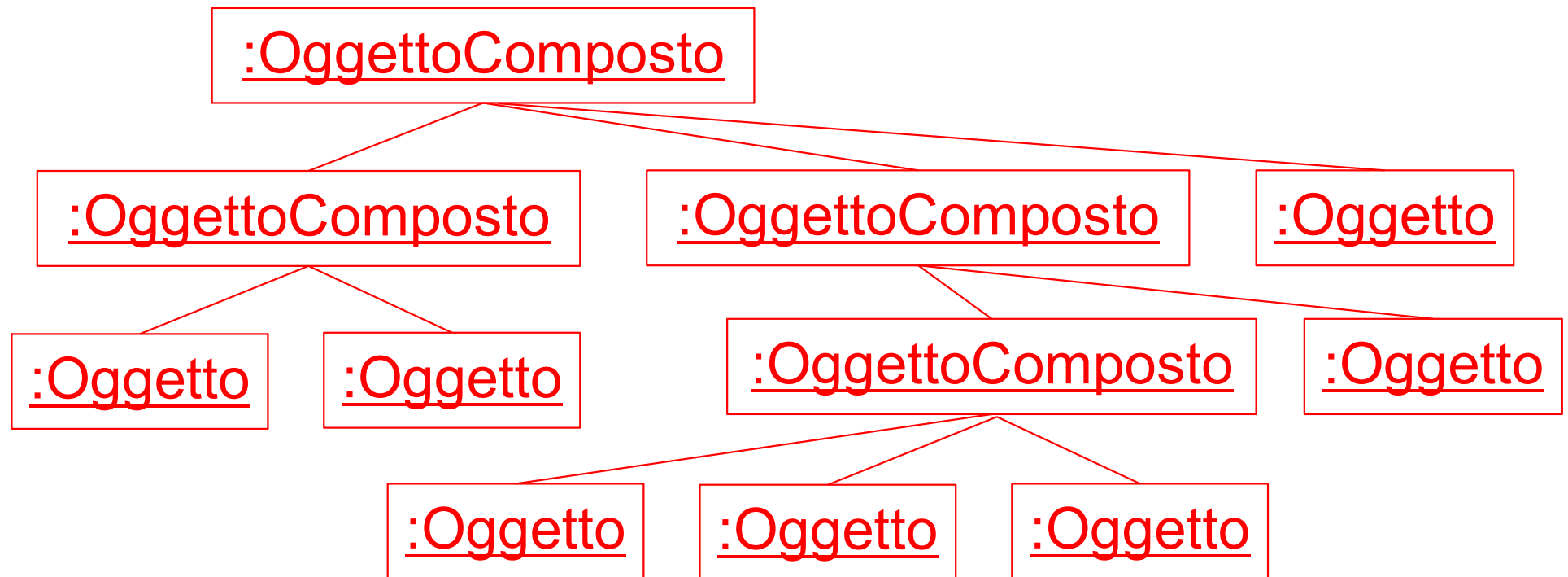
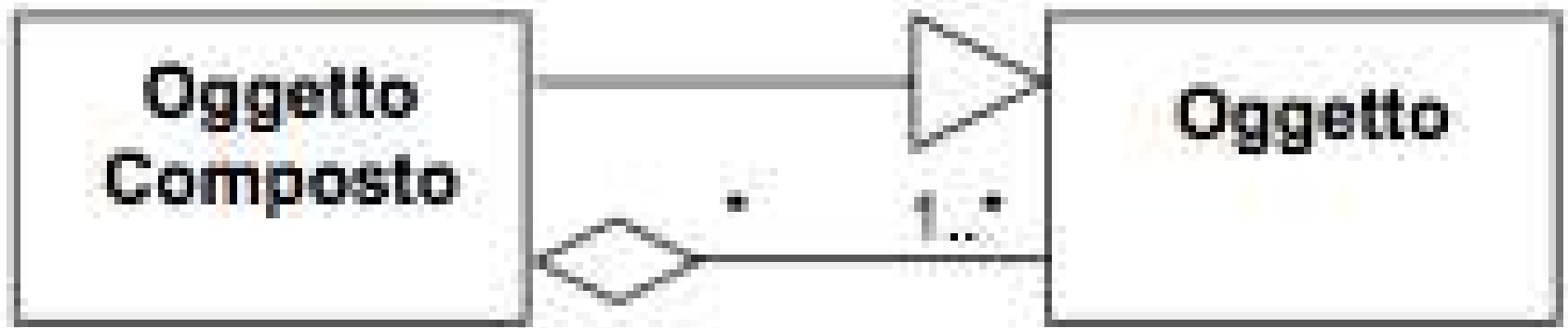
- I parallelepipedo hanno sempre le facce parallele ai piani cartesiani
- Il triangolo di base dei tetraedri è sempre equilatero e (posto su un piano) parallelo a uno dei piani identificati dagli assi cartesiani

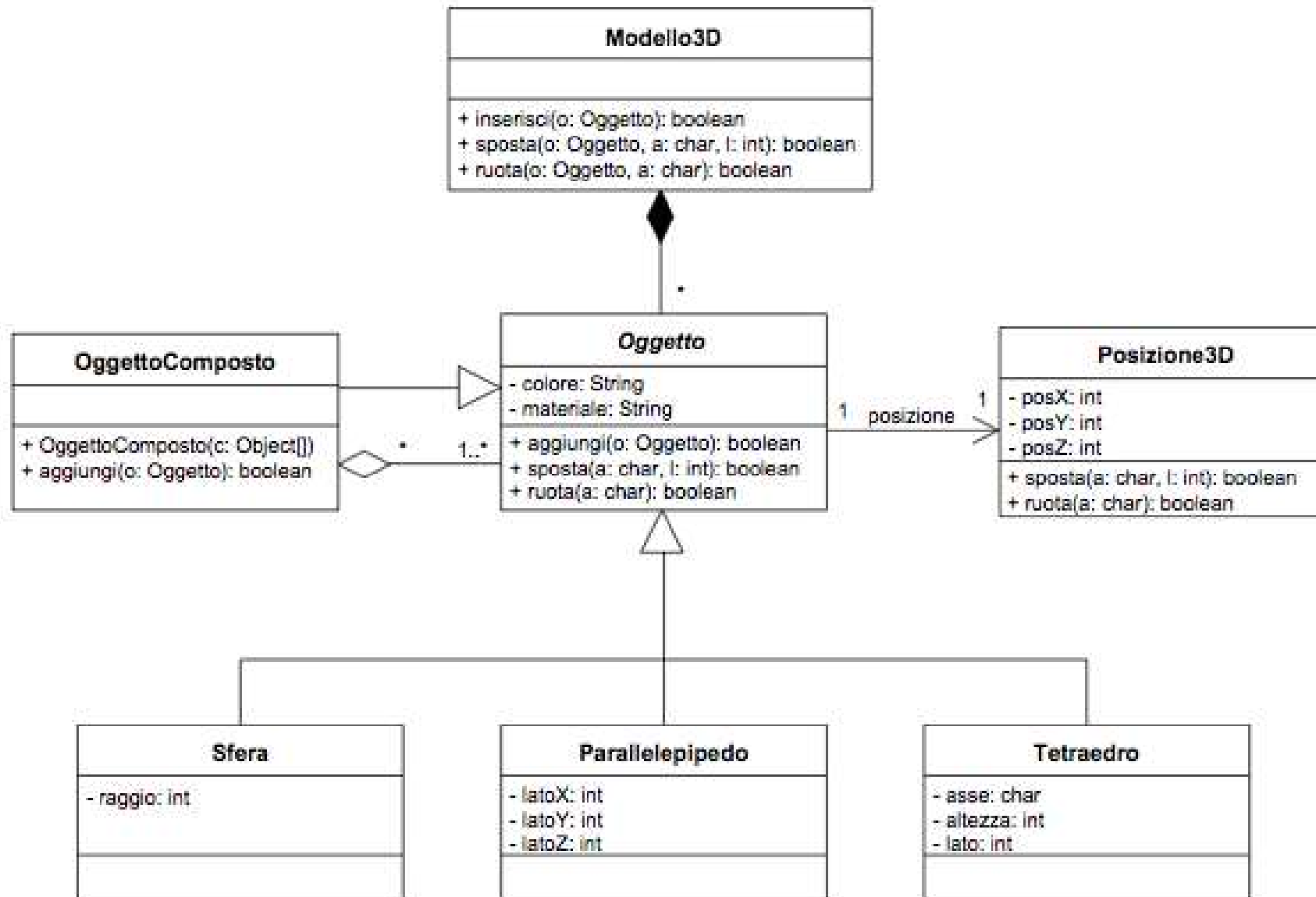
Vincoli aggiunti implicitamente dal libro di testo

- Gli spostamenti degli oggetti possono avvenire solo parallelamente a un asse
- Le rotazioni possono essere solo di 180°

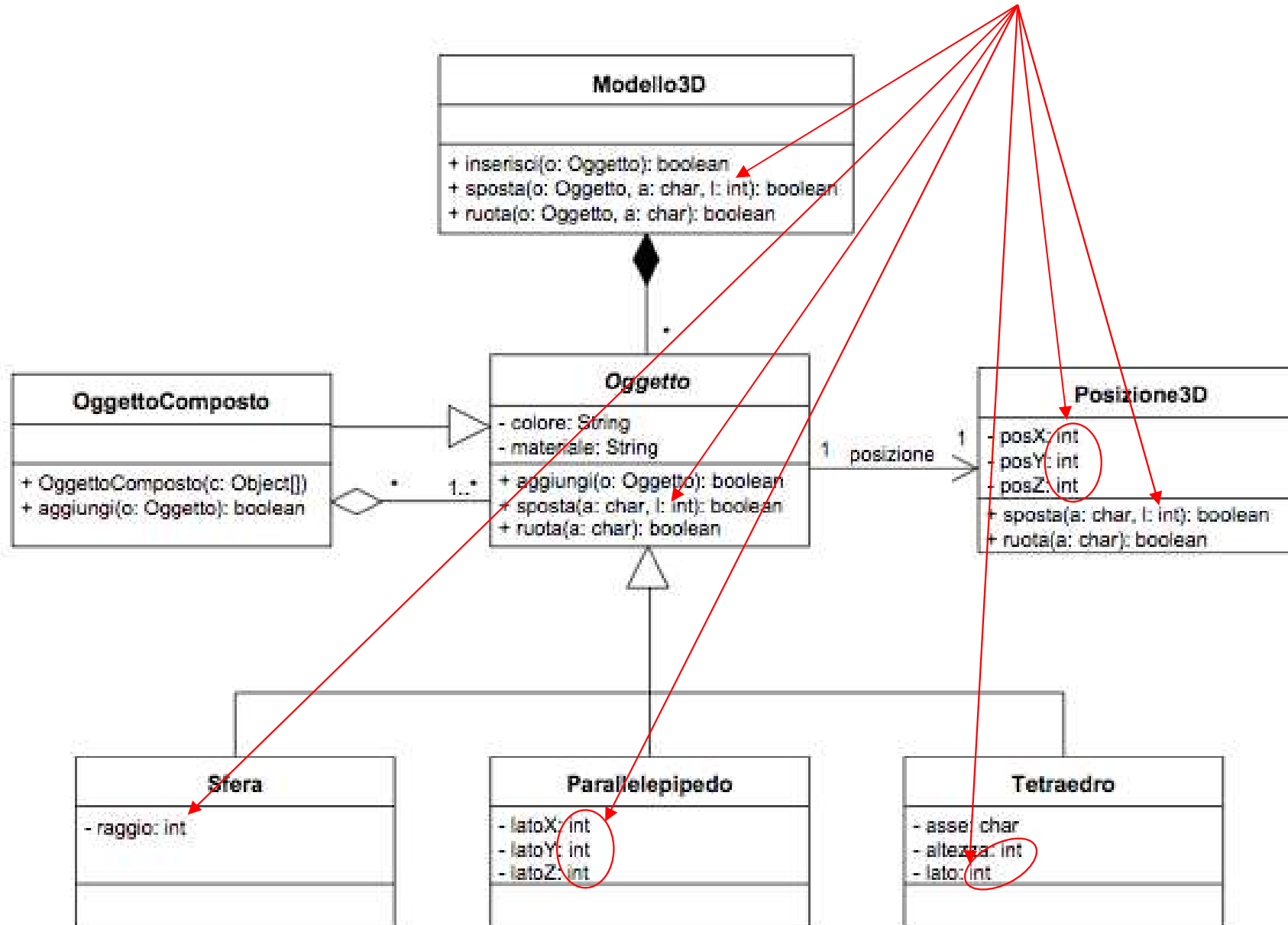


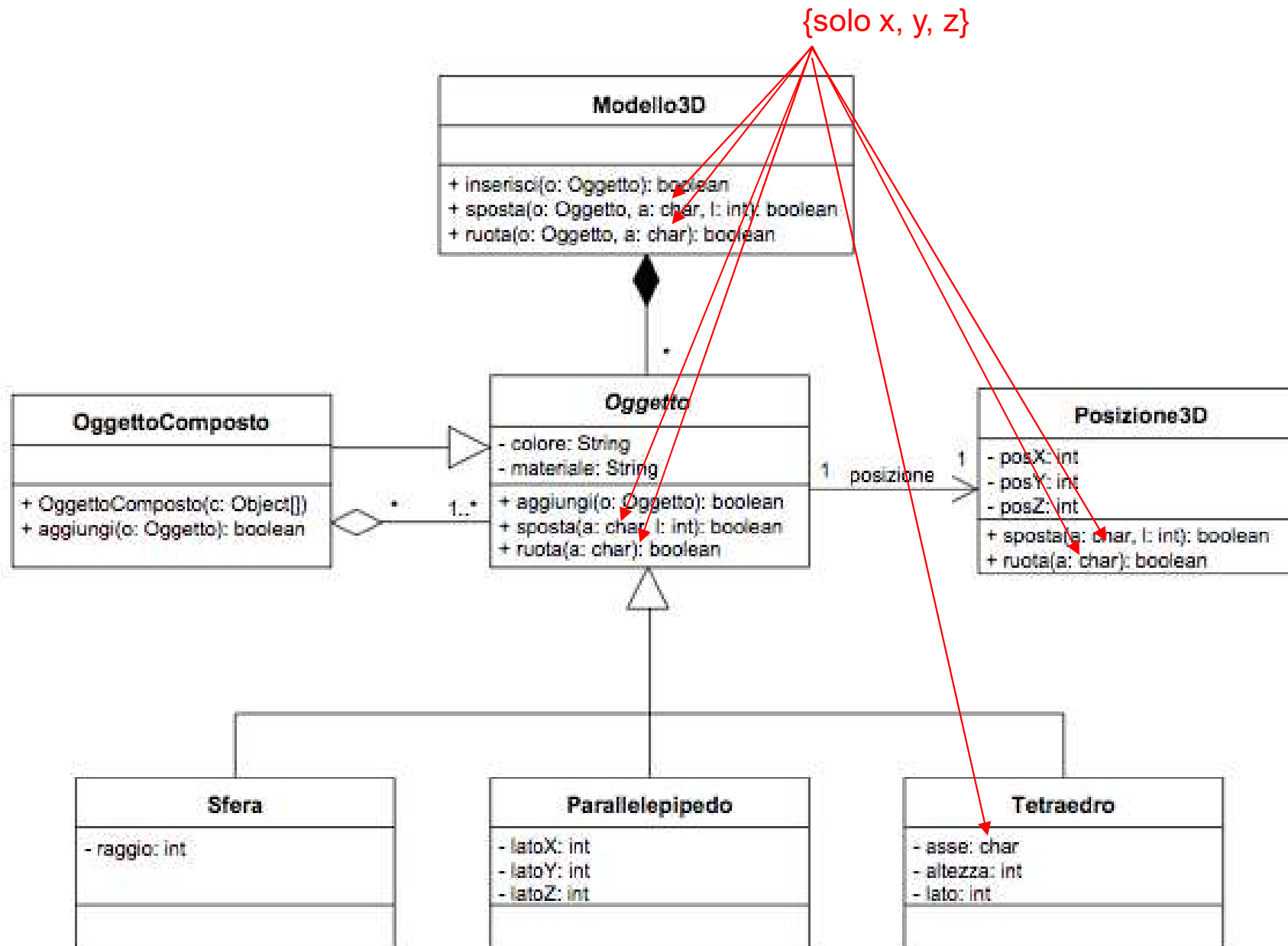






tipo ragionevole?



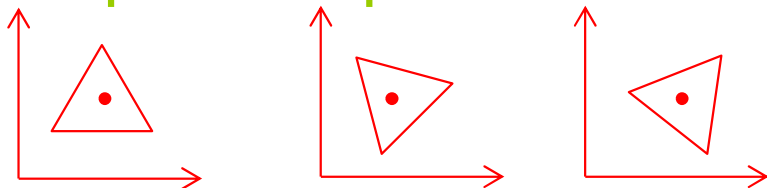


Manca qualcosa?

Sì, perché il tetraedro non è univocamente definito; è necessario aggiungere questo vincolo:

In ogni tetraedro

- la proiezione del quarto vertice è il centro del triangolo di base
- il triangolo di base ha un lato parallelo a un asse dato e tale lato ha una distanza da suddetto asse minore rispetto a quella del vertice opposto



Di conseguenza

Un'istanza di Tetraedro necessita dell'indicazione di due assi (uno parallelo all'altezza, l'altro parallelo al lato del triangolo che costituisce la base)

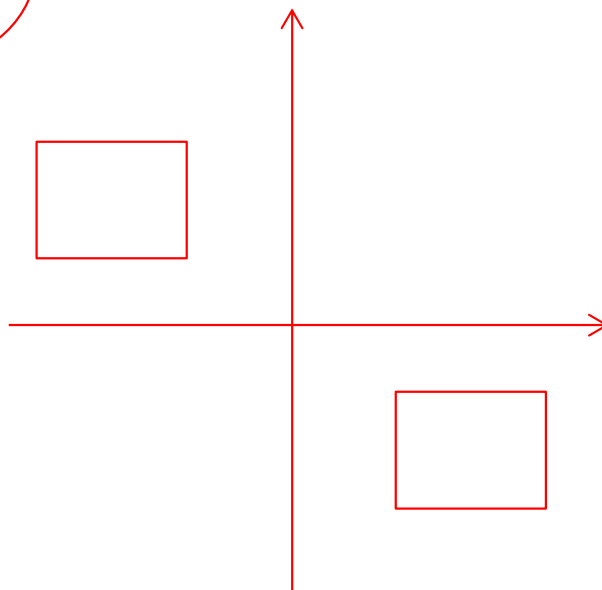
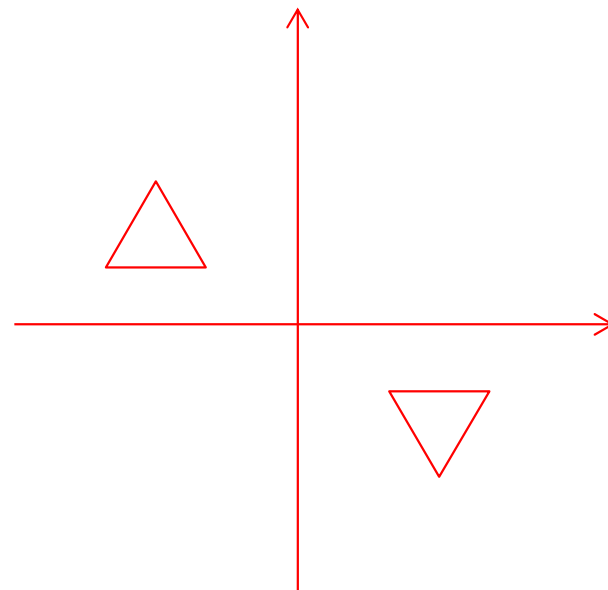
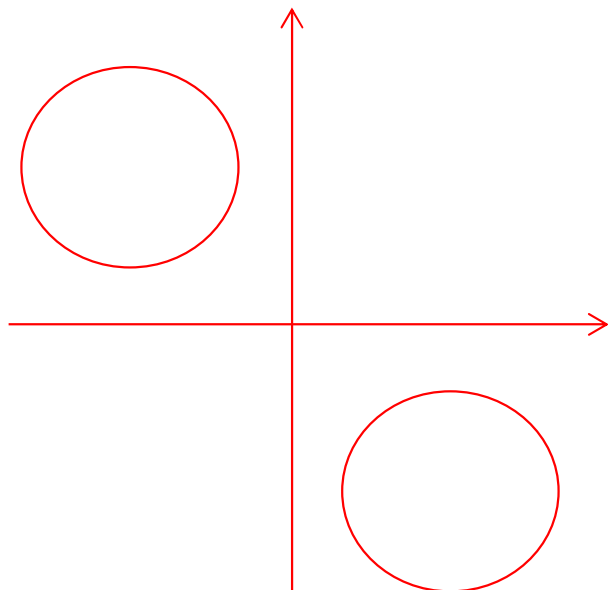


È necessario aumentare a 2 la molteplicità dell'attributo **asse** della classe **Tetraedro**

Rotazione

- È necessario stabilire cosa si intenda: rotazione intorno a un asse di un sistema di assi cartesiani fisso oppure rotazione rispetto a un asse di un sistema di assi cartesiani solidale con l'oggetto tridimensionale (cioè rotazione dell'oggetto su se stesso)?
- Sembra che valga la prima interpretazione, dal momento che la rotazione ha effetto solo sulla posizione del solido
- Il fatto che la rotazione abbia effetto solo sulla posizione del solido induce a ritenere che per un parallelepipedo tale posizione sia il centro del solido e per il tetraedro sia il vertice non appartenente alla base

Rotazione: interpretazione



Inoltre

Perché a un oggetto composto corrisponde una posizione tridimensionale? Il costruttore `OggettoComposto` non riceve alcun parametro al fine di inizializzare la stessa



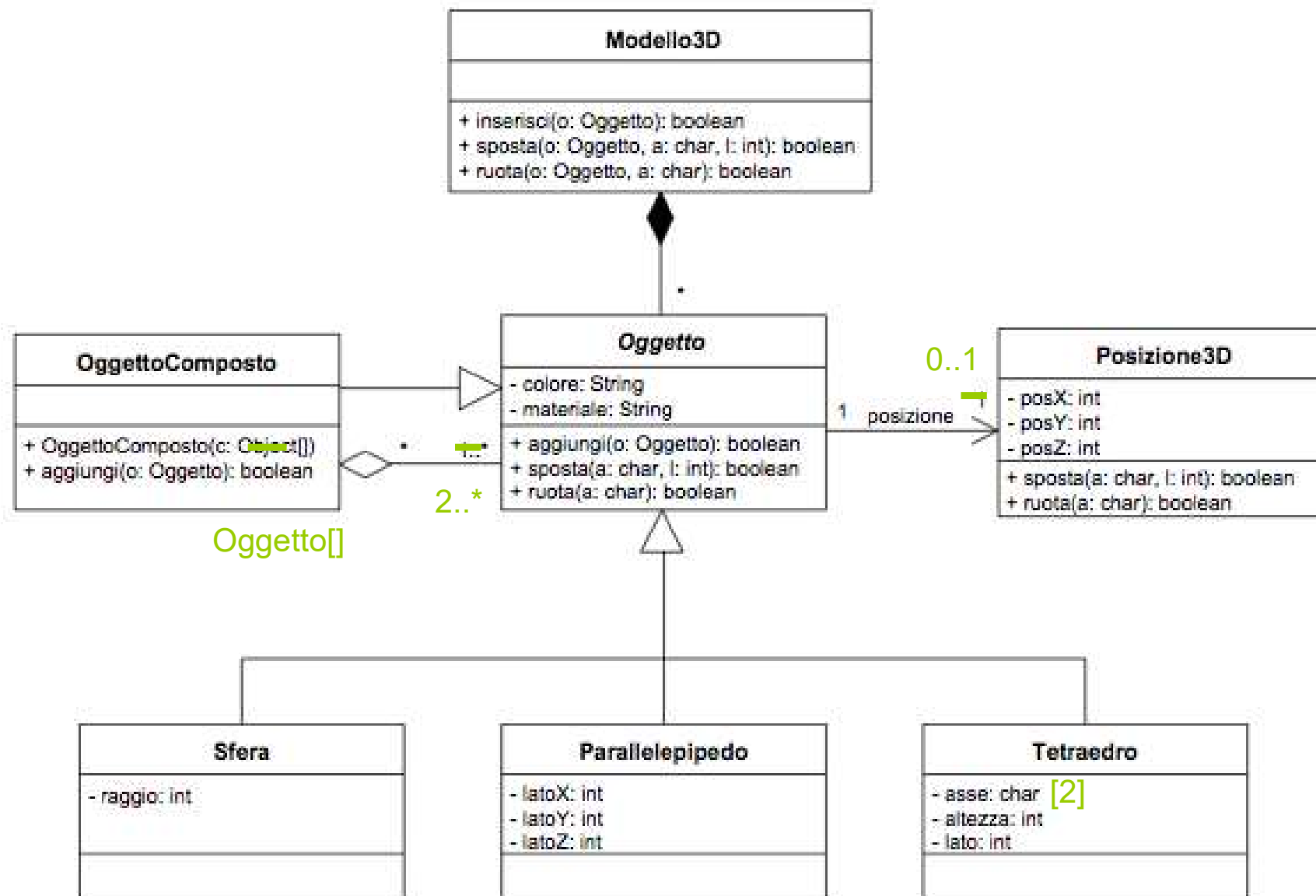
È opportuno ridurre a 0 l'estremo inferiore della molteplicità dell'associazione `posizione` fra le classi *Oggetto* e `Posizione3D`

Analogamente

Perché un oggetto composto deve ereditare gli attributi **colore** e **materiale**?
Il costruttore OggettoComposto non riceve alcun parametro al fine di inizializzare gli stessi (**ma il loro valore potrebbe essere scelto automaticamente**)

Infine

L'operazione **aggiungi** della classe astratta *Oggetto* viene sovrascritta nella classe concreta *OggettoComposto*. Anche le operazioni **sposta** e **ruota** devono essere sovrascritte: perché invece non compaiono in *OggettoComposto*?



3. Rivendita di auto usate

Un automezzo viene identificato dalla targa, dal numero di telaio e da un certo numero di caratteristiche specifiche, quali il colore, la cilindrata, il tipo di carburante e gli optional.

Rivendita di auto usate

Ogni automezzo è caratterizzato da una “carta di identità” che definisce l’anno di immatricolazione, il numero di chilometri e la data dell’ultima revisione. Il sistema gestisce anche camion e van, che si differenziano dalle automobili per la capacità di carico (quintali o persone).

Rivendita di auto usate (cont.)

Il sistema cataloga anche i clienti, con le solite caratteristiche: nome, cognome, indirizzo e codice fiscale.

Un cliente può stipulare uno o più contratti per acquistare uno o più automezzi.

Rivendita di auto usate (cont.)

Ogni contratto deve avere una data di stipula, un ammontare, una data di inizio validità ed eventuali dilazioni di pagamento pattuite tra le parti.

Il sistema deve anche gestire lo storico dei diversi automezzi, cioè la storia del mezzo che contiene tutti i passaggi di proprietà noti al rivenditore.

3. Rivendita di auto usate

Un automezzo viene identificato dalla targa, dal numero di telaio e da un certo numero di caratteristiche specifiche, quali il colore, la cilindrata, il tipo di carburante e gli optional.

Rivendita di auto usate

Ogni automezzo è caratterizzato da una “**carta di identità**” che definisce l’anno di immatricolazione, il numero di chilometri e la data dell’ultima revisione. Il sistema gestisce anche **camion** e **van**, che si differenziano dalle automobili per la capacità di carico (quintali o persone).

Rivendita di auto usate (cont.)

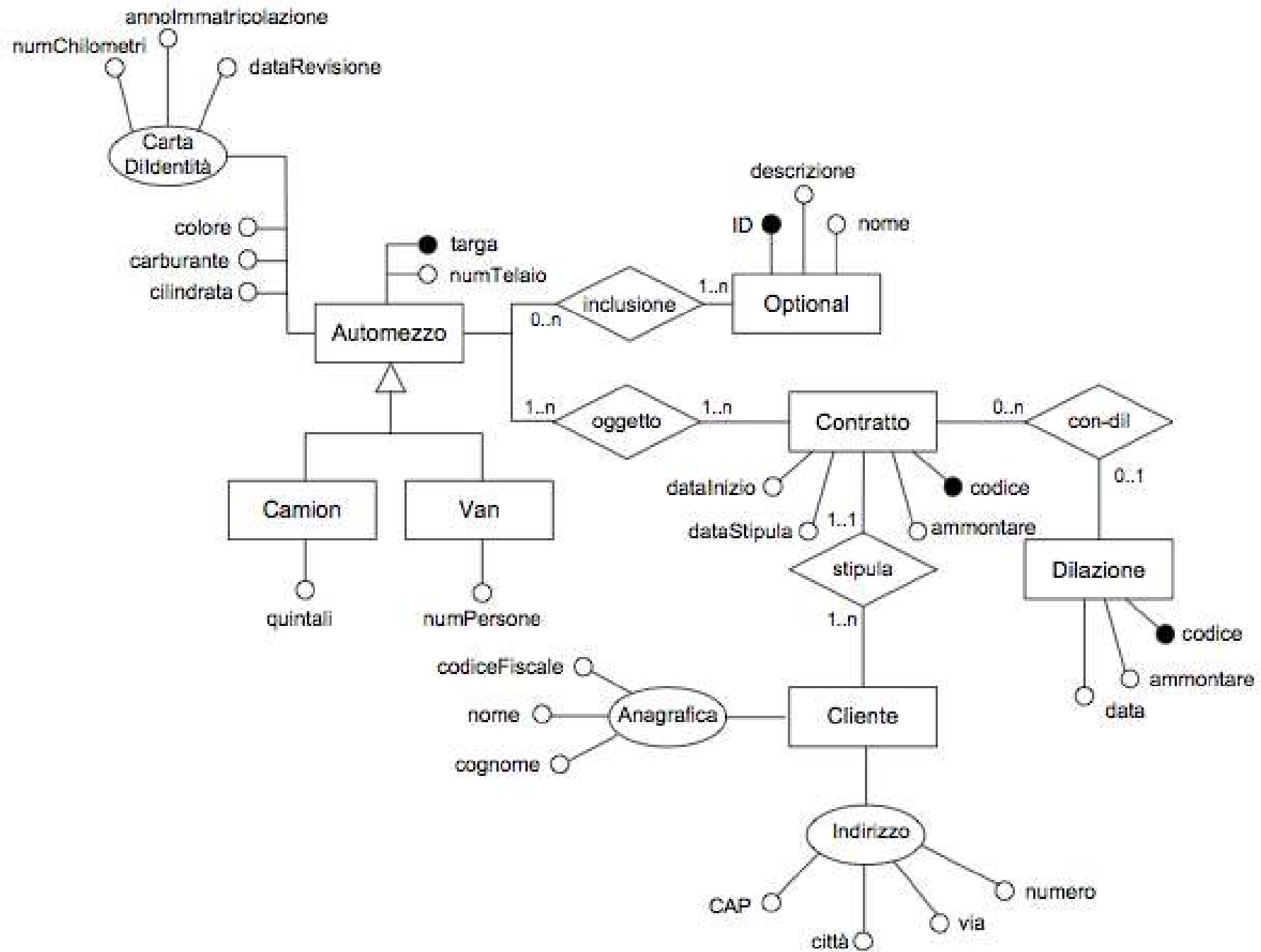
Il sistema cataloga anche i **clienti**, con le solite caratteristiche: **nome**, **cognome**, **indirizzo** e **codice fiscale**.

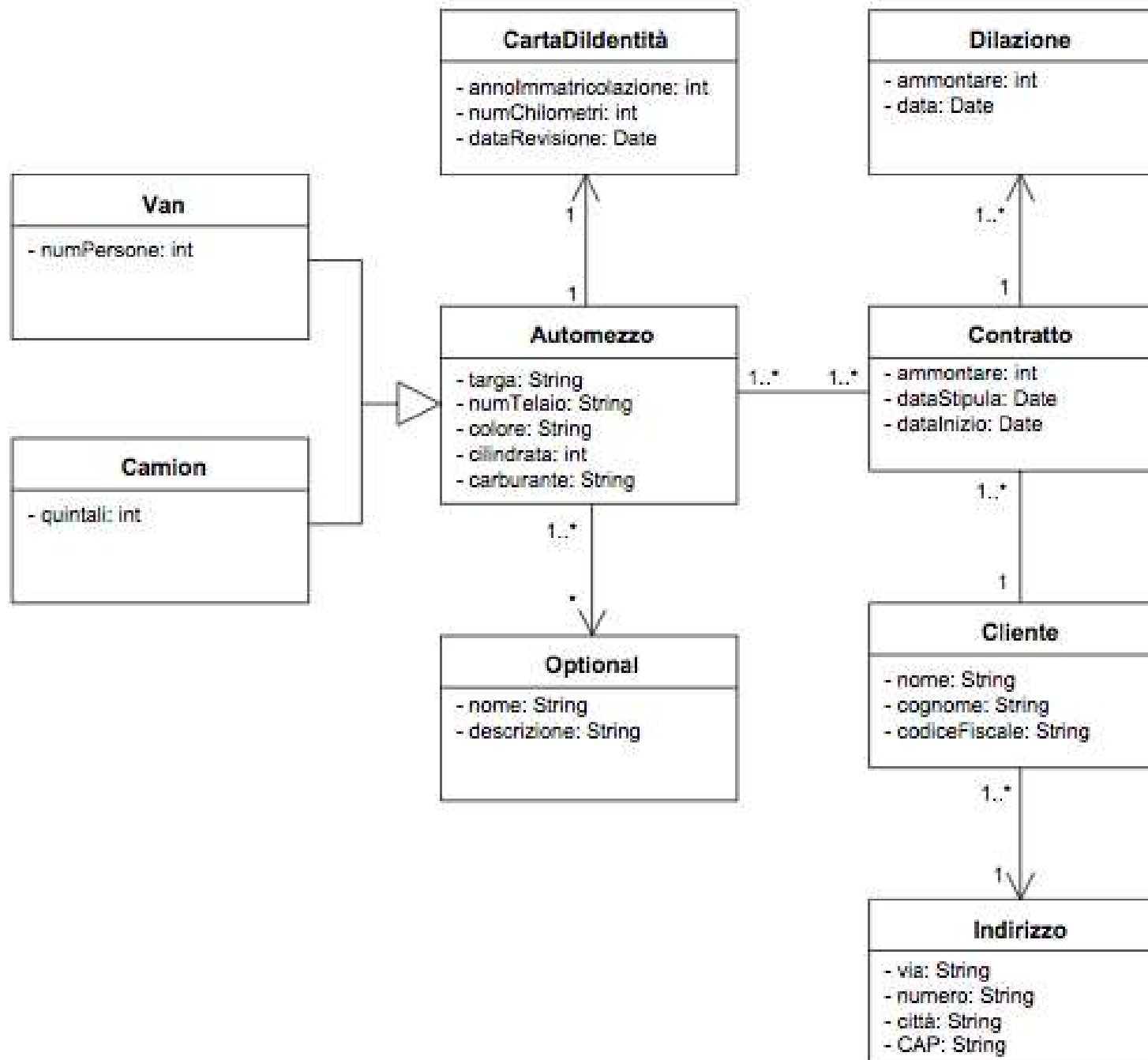
Un cliente può stipulare uno o più contratti per acquistare uno o più automezzi.

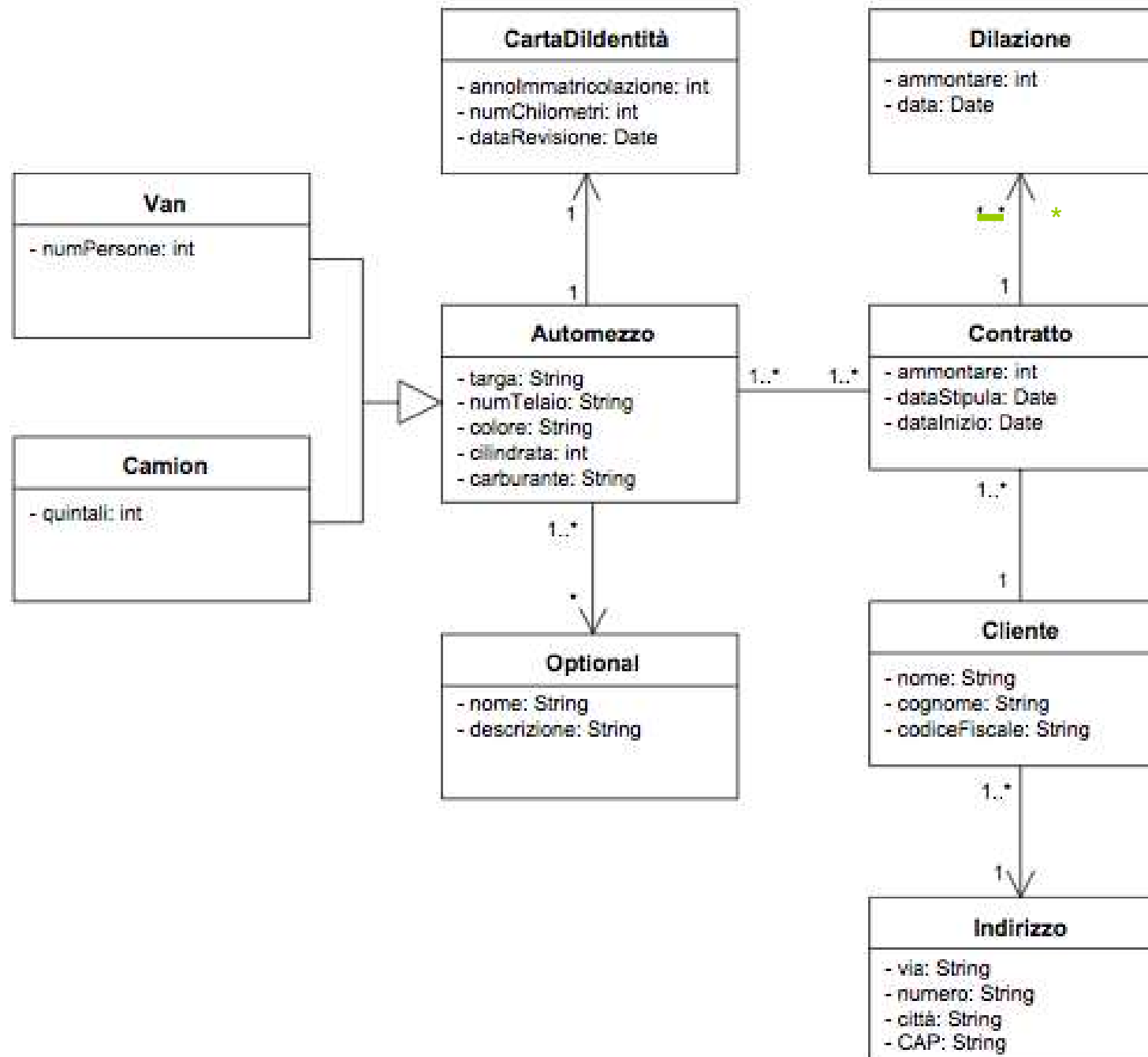
Rivendita di auto usate (cont.)

Ogni **contratto** deve avere una **data di stipula**, un **ammontare**, una **data di inizio** validità ed eventuali **dilazioni** di pagamento pattuite tra le parti.

Il sistema deve anche gestire lo storico dei diversi automezzi, cioè la storia del mezzo che contiene tutti i passaggi di proprietà noti al rivenditore.







Componenti

- Cosa sia precisamente un componente è a oggi argomento di intenso dibattito
- I componenti rappresentano (anche) pezzi di sw che possono essere acquisiti e aggiornati in modo indipendente

In UML \exists le due sottostanti rappresentazioni alternative di un componente

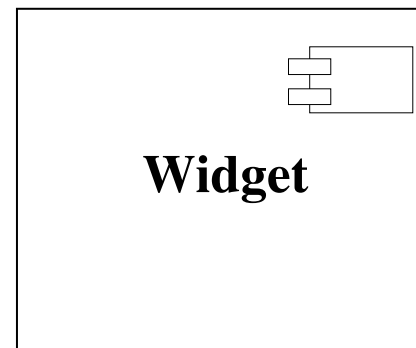
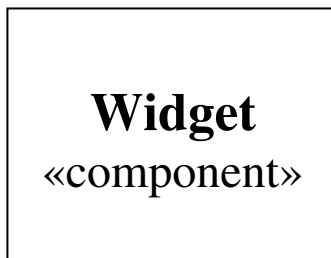


Diagramma dei componenti: sintassi

- I collegamenti fra componenti sono basati sulle interfacce implementate e richieste (notazione “pallina e socket”)
- Per mostrare la composizione interna di un componente si può usare un diagramma di struttura composita, in cui ogni parte può essere a sua volta un componente

Esempio

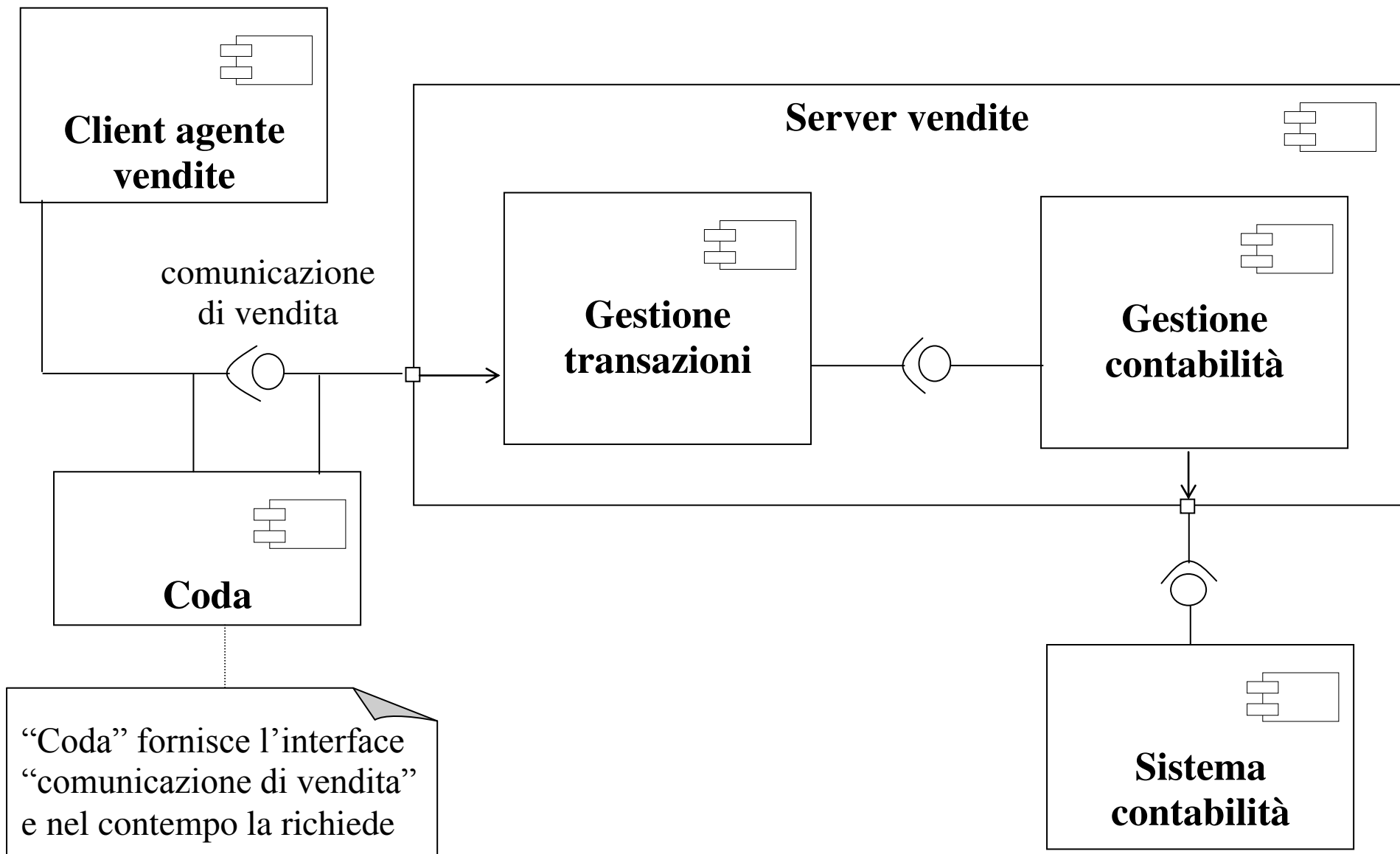


Diagramma dei componenti

Si usa

- quando si sta suddividendo il sistema in componenti (progettazione)
- per documentare le relazioni fra componenti e interfacce
- per mostrare la struttura interna dei singoli componenti a un livello di astrazione più basso

Fowler consiglia di diffidare dei componenti di grana eccessivamente fine: troppi componenti sono difficili da gestire, specialmente quando di questi vengono prodotte più versioni

Diagrammi di stato

- Corrispondono alle statechart di Harel (1987)
- Ogni diagramma è associato a una classe e serve per descrivere il comportamento di un oggetto, istanza di quella classe, per la durata del suo ciclo di vita (attraverso più casi d'uso)
- Sono molto usati per la progettazione di sistemi in tempo reale
- Si suggerisce di adottarli per descrivere le classi che hanno una logica interna interessante e complessa, tipicamente quelle di controllo e quelle che realizzano le interfacce grafiche, così da comprenderne il funzionamento

Diagramma di stato

Elementi	Sintassi	Semantica
Punto di partenza (pseudostato iniziale)	Punto nero pieno ●, elemento obbligatorio (sostituibile con uno pseudostato di storia), da cui esce una freccia di transizione di stato	Segnala il punto in cui inizia il ciclo di vita dell'oggetto Non è un vero stato: il vero stato iniziale è quello puntato dalla freccia
Punto di arrivo (stato finale)	⦿, elemento facoltativo in cui entra una freccia di transizione di stato	Segnala la fine dell'esecuzione relativa all'oggetto (cioè la cancellazione dello stesso) o la fine di un comportamento indipendente entro uno stato concorrente

Diagramma di stato (cont.)

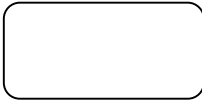
Elementi	Sintassi	Semantica
Stato	<p>Rettangolo con angoli smussati, eventualmente suddiviso orizzontalmente in due parti da una linea continua</p> 	<p>Situazione in cui l'oggetto svolge un'attività e/o è in attesa di un evento</p> <p>Astrazione che individua diversi comportamenti dell'oggetto al verificarsi degli eventi</p>
Nome di uno stato/superstato	<p><i>nome_(super)stato</i>, elemento obbligatorio scritto in grassetto entro il riquadro dello stato; se tale riquadro è diviso in due parti, deve essere scritto – da solo – nella parte superiore</p>	<p>Nome unico dello stato: stati contraddistinti dallo stesso nome in diagrammi distinti sono indistinguibili, cioè rappresentano lo stesso stato</p>

Diagramma di stato (cont.)


Elementi	Sintassi	Semantica
Attività di uno stato (do-activity)	<i>do/ attività</i> , elemento opzionale scritto entro lo stato (il quale è pertanto uno stato di attività)	Processo, interrompibile da eventi, eseguito quando l'oggetto si trova nello stato Se il processo si conclude, l'oggetto resta nello stato fino a che non scatta una transizione
Transizione	Freccia con punta biforcuta e linea continua che si diparte dallo stato sorgente e punta allo stato destinazione (che può coincidere con il sorgente, nel qual caso si parla di autotransizione o autoanello) 	Cambiamento di stato

Diagramma di stato (cont.)

Elementi	Sintassi	Semantica
Etichetta di una transizione	<p>Tre elementi, <i>evento [condizione] / attività</i>, tutti opzionali, scritti sopra la freccia che indica la transizione, dove <i>condizione</i> (o <i>guardia</i>) è un'espressione logica</p> <p>Una condizione mancante equivale a [true]</p> <p>Date tutte le transizioni uscenti da un medesimo stato, ogni coppia <i>evento [condizione]</i> che le contraddistingue è unica e le diverse condizioni inerenti allo stesso evento sono mutuamente esclusive (ciò garantisce che uscendo da uno stato si possa compiere una e una sola transizione)</p>	<ul style="list-style-type: none"> • <i>evento</i> (o <i>trigger</i>) è il nome dello stimolo esterno (anche composto/parametrico) che, nel caso <i>condizione</i> (se presente) sia vera, fa scattare immediatamente la transizione, indipendentemente dal fatto che la (eventuale) do-activity associata allo stato da cui la transizione è uscente sia conclusa o meno • Se <i>evento</i> manca, la transizione è abilitata a scattare non appena viene completata la (eventuale) do-activity associata allo stato da cui la transizione è uscente, ma scatta effettivamente solo se (o quando) <i>condizione</i> è vera • <i>attività</i> è un processo atomico associato alla transizione eseguito prima di entrare nello stato destinazione della transizione stessa • Se, quando l'oggetto è in un certo stato, si verifica un evento per il quale non c'è nessuna transizione uscente, l'evento è ignorato

Attività interne

Elementi	Sintassi	Semantica
Evento <i>entry</i> (o attività interna di entrata)	<i>entry</i> / <i>attività</i> , elemento opzionale scritto entro uno stato	L' <i>attività</i> è atomica e viene eseguita ogniqualevolta si entra nello stato, prima dell'eventuale attività associata allo stato stesso (do-activity)
Evento <i>exit</i> (o attività interna di uscita)	<i>exit</i> / <i>attività</i> , elemento opzionale scritto entro uno stato	L' <i>attività</i> è atomica e viene eseguita ogniqualevolta si esce dallo stato
Attività interna	Come l'etichetta di una transizione ma scritta entro il box di uno stato	Processo atomico È simile a un'autotransizione ma si distingue da questa perché non fa scattare le attività di entrata e di uscita

L'esempio di un impianto di condizionamento

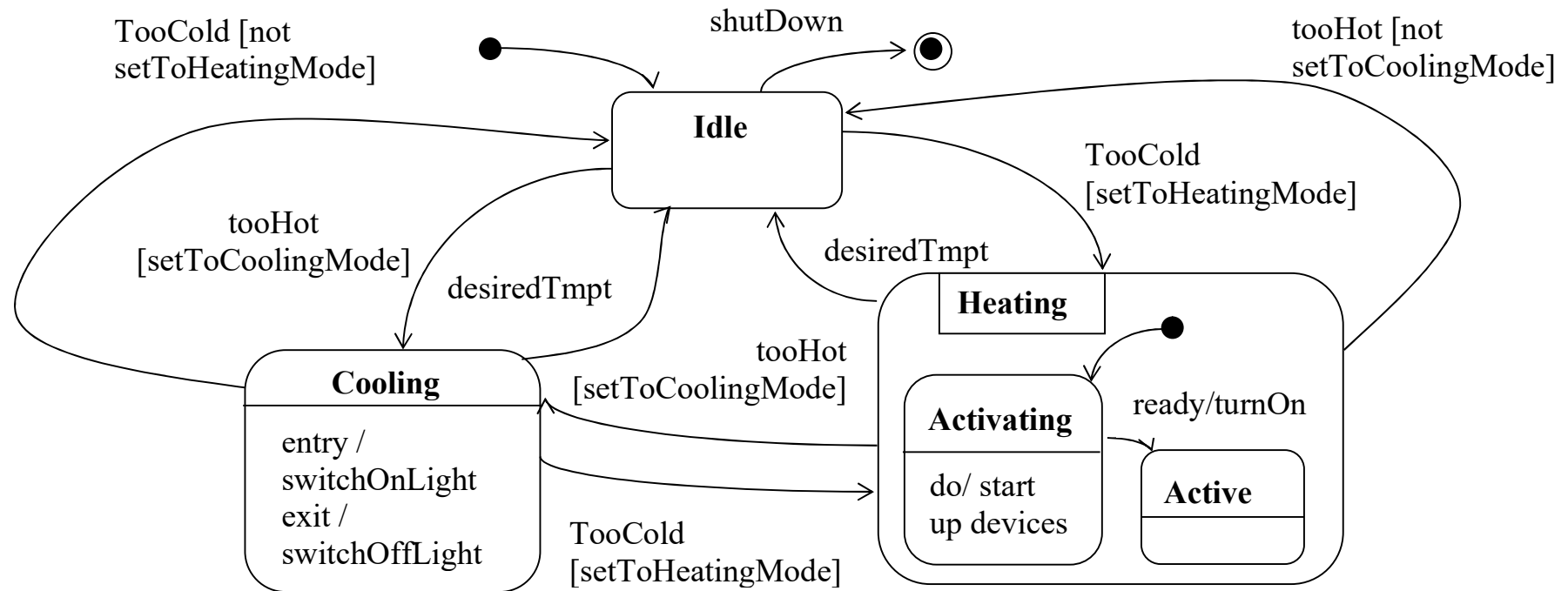


Diagramma di stato (cont.)

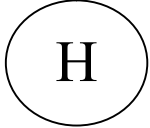
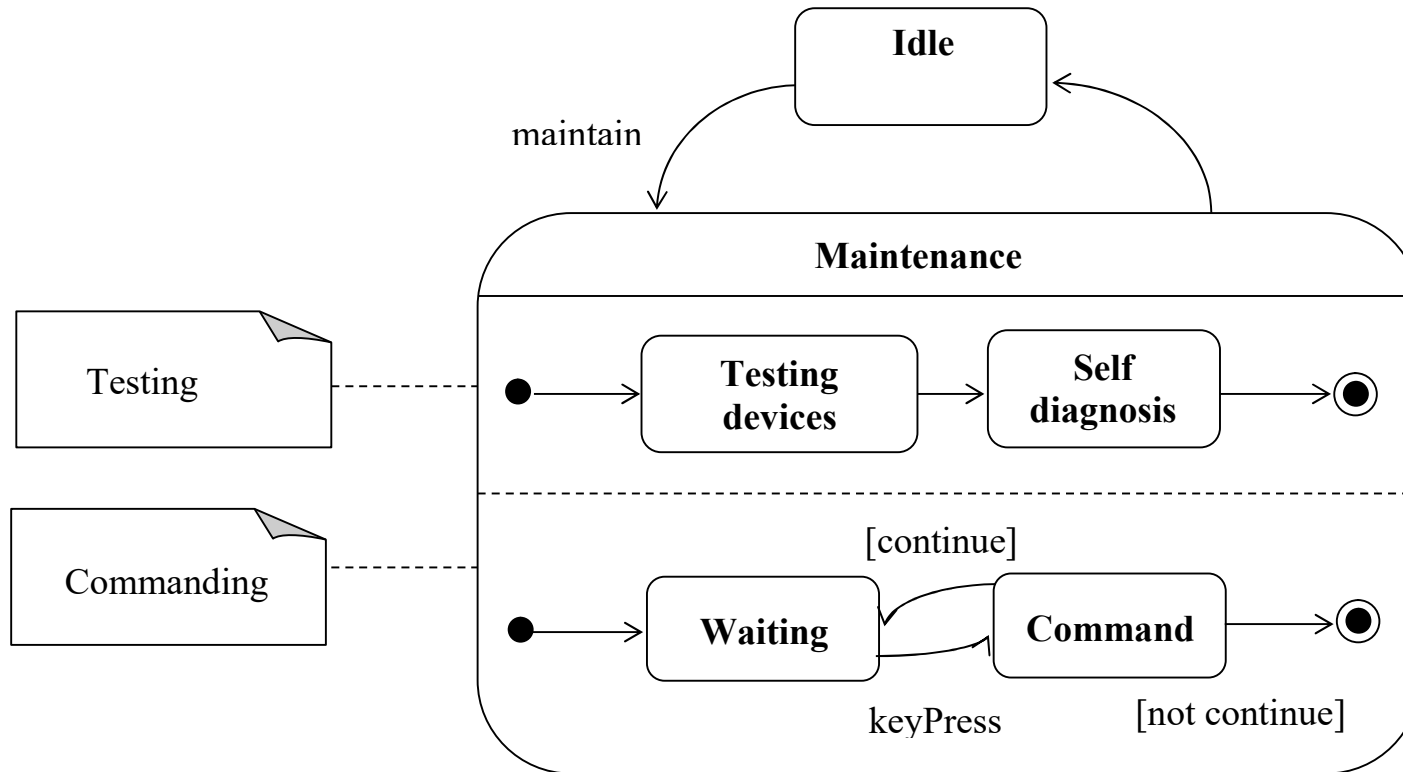
Elementi	Sintassi	Semantica
Evento speciale <i>after</i>	<i>after (lasso di tempo)</i> , scritto laddove è previsto il nome di un evento	Indica un evento generato dopo il <i>lasso di tempo</i> fissato, ad es. <i>after(2 sec)</i>
Evento speciale <i>when</i>	<i>when (condizione)</i> , scritto laddove è previsto il nome di un evento	Indica un evento che si verifica quando la <i>condizione</i> è vera
Pseudostato di storia	Cerchio contenente la lettera H (per <i>history</i>) da cui esce la freccia di una transizione di stato, diretta al riquadro di uno stato 	Sostituisce lo pseudostato iniziale Al momento dell'avvio del suo ciclo di vita, l'oggetto si trova nell'ultimo stato raggiunto nel ciclo di vita precedente La destinazione della freccia è lo stato iniziale dell'oggetto al primo ciclo di vita, cioè quando non c'è una storia passata

Diagramma di stato (cont.)

Elementi	Sintassi	Semantica
Super-stato	<ul style="list-style-type: none"> • Box contenente un diagramma degli stati, privo di punto di arrivo, ed eventuali attività interne • Possono esistere transizioni entranti/uscenti in/da uno stato interno che entrano/escono anche nel/dal superstato 	<ul style="list-style-type: none"> • Ogni transizione entrante nel superstato entra nel punto di partenza del superstato (o nell'ultimo stato lasciato, se il punto di partenza è uno pseudostato di storia) • Ogni transizione uscente dal superstato esce da qualsiasi stato interno al superstato • Ogni stato interno al superstato condivide le attività interne indicate nel superstato
Stato concorrente	Box contenente un diagramma degli stati concorrente, cioè più diagrammi degli stati separati reciprocamente da una linea orizzontale tratteggiata	<ul style="list-style-type: none"> • Rappresentazione di processi concorrenti • Lo stato di un diagramma degli stati concorrente è dato dalla combinazione di più stati, uno per ogni diagramma che rappresenta un processo eseguito in parallelo

Un diagramma di stato concorrente



Modelli di processo

Ciclo di vita di un prodotto sw = tutte le fasi che accompagnano tale prodotto dal concepimento dell'idea dello stesso fino al suo ritiro

Modello di processo = tentativo di organizzare il ciclo di vita del sw

- definendo le attività coinvolte nella produzione del sw
- ordinando tali attività e le loro relazioni (stabilendo cioè il flusso di esecuzione)

“determinare l'ordine delle fasi coinvolte nello sviluppo e nell'evoluzione del sw e stabilire i criteri di passaggio da una fase alla successiva. Ciò comprende criteri di completamento della fase corrente più criteri di ingresso nella successiva. Quindi un modello di processo affronta le seguenti domande relative a un progetto sw:

Che facciamo adesso?

Fino a quando continuiamo a farlo?”

(Boehm, 1988)

Obiettivi di un modello di processo

- introdurre disciplina
- standardizzare
- dominare la complessità
- migliorare verificabilità, manutenibilità, riusabilità, comprensibilità, produttività, visibilità e tempestività del processo
- aumentare la capacità di prevedere tempi e costi
- rendere il processo più facilmente automatizzabile
- migliorare la qualità dei prodotti

Code&Fix

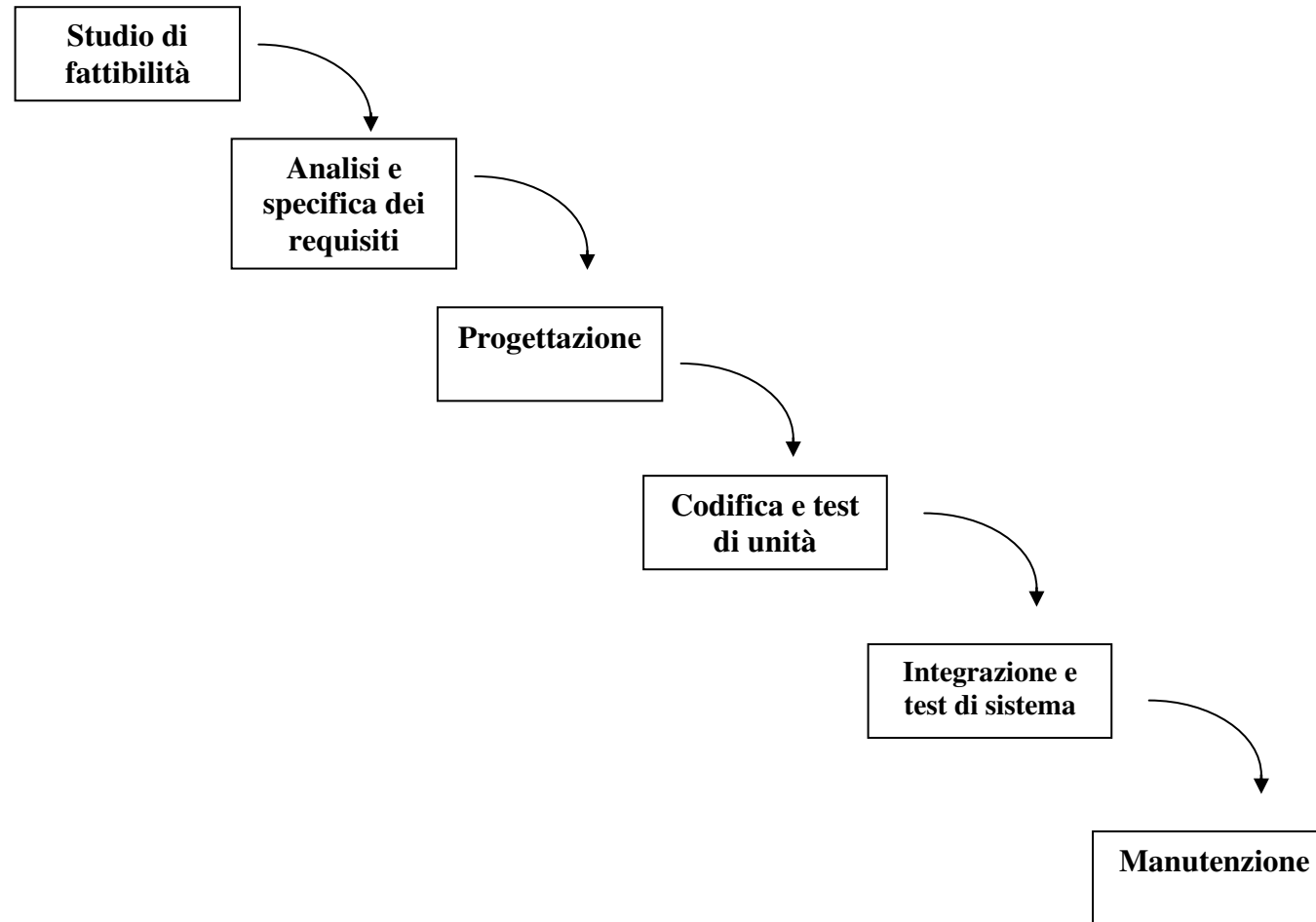
È l'approccio più antico, secondo cui

- si scrive il codice
- lo si aggiusta per eliminare gli errori che sono stati scoperti, per migliorare le funzionalità esistenti e/o per aggiungere nuove caratteristiche

Carenze:

- impossibile fare previsioni
- processo ingestibile

Modello a cascata (Royce 1970)



Modello a cascata: fasi

Inventato nei tardi anni '50 per grandi sistemi di difesa aerea, divenuto popolare negli anni '70

Fasi:

- 1) Studio di fattibilità: definizione preliminare del problema, valutazione a priori di costi e benefici; obiettivo: stabilire se lo sviluppo debba essere avviato, evidenziare le risorse disponibili per il progetto, elencare e comparare le alternative
- 2) Analisi e specifica dei requisiti: analisi completa del problema dell'utente e della sua realtà applicativa al fine di specificare le caratteristiche di qualità e i requisiti funzionali dell'applicazione (*cosa* il sistema deve fornire, non *come*). I risultati di tale analisi (spesso incompleti/inconsistenti/ambigui) devono essere sottoscritti dal committente
- 3) Progettazione: definizione dell'architettura sw

Modello a cascata: fasi (cont.)

Fasi:

- 4) Codifica e test di unità: programmazione (distinzione sfumata rispetto alla progettazione) + test per verificare il soddisfacimento delle specifiche di progetto
- 5) Integrazione e test di sistema: collaudo dell'intero sistema + (opzionalmente) alfa test (rilascio entro l'organizzazione del produttore) e beta test (rilascio a pochi utenti selezionati)
- 6) Manutenzione

Modello a cascata: pro ...

Se si adotta il modello a cascata, è necessario definire con precisione contenuti e struttura dei semilavorati e, in fase di pianificazione, le scadenze entro cui devono essere prodotti e superare i controlli di qualità

Pro:

- Le fasi indirizzano l'attività del progettista e consentono il controllo dello svolgimento del progetto
- Rimanda l'implementazione a dopo che gli obiettivi sono stati compresi

... e contro

Contro:

- Congela i requisiti
- Le fasi non sono formalmente definite, né passibili di svolgimento o controllo automatici
- I ricicli (retroazioni) sono nascosti
- È difficile raccogliere i requisiti una volta per tutte
- Una sola data di consegna
- Non include la gestione dei cambiamenti → manutenzione ed evoluzione sono di difficile previsione ed attuazione
- Stati bloccanti

Modello a cascata: rischi

Rischi:

- Individuare scelte ottimali solo per l'applicazione attuale, senza pensare all'evoluzione futura dell'applicazione
- Non pianificare l'attività di manutenzione ed eseguirla solo sul codice, non sugli altri semilavorati

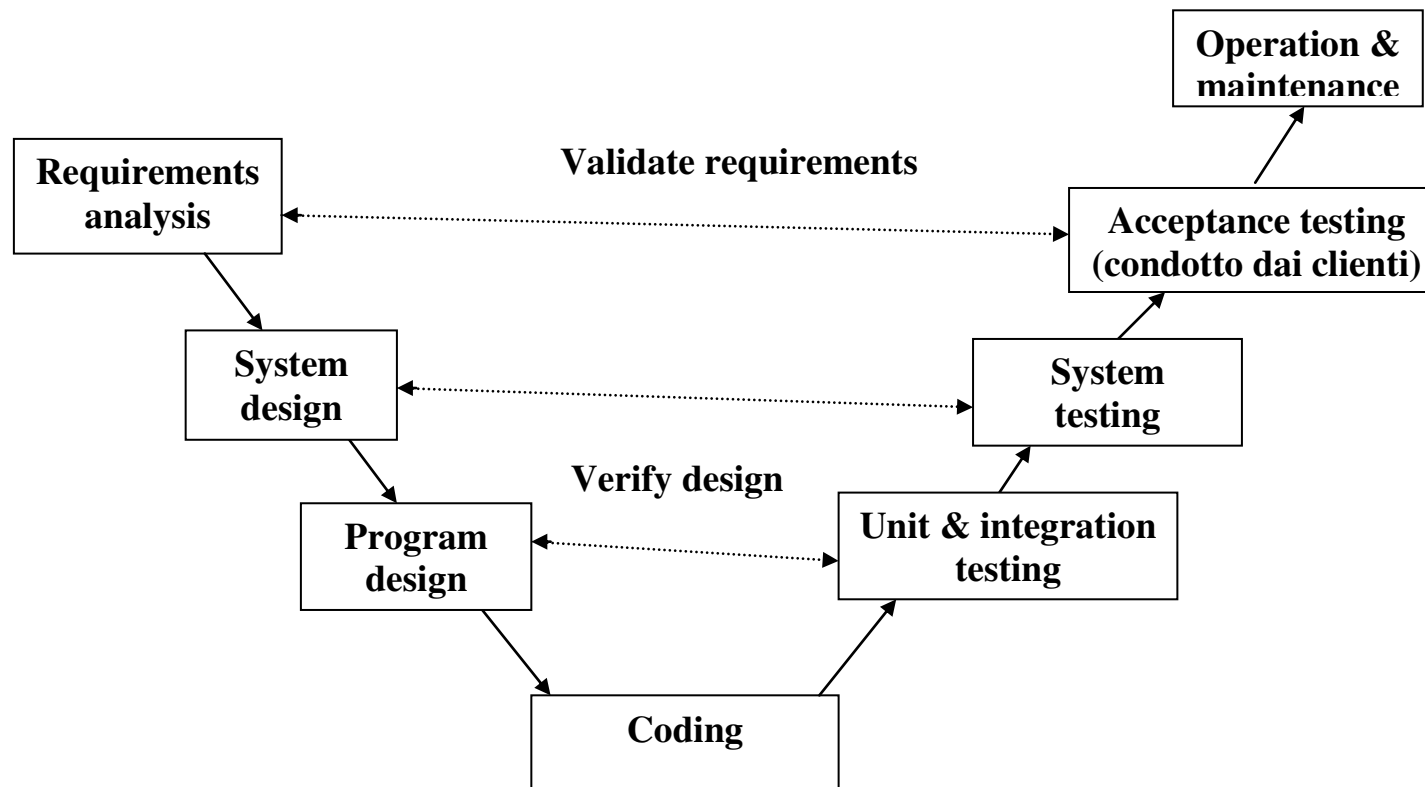
Tutto ciò è miope e si paga a caro prezzo →

Reingegnerizzazione: riportare sw destrutturato e non documentato in uno stato dal quale si possa ripartire per una manutenzione sistematica

Conclusione:

se il maggiore rischio è l'affidabilità dell'applicazione, mentre i requisiti sono estremamente stabili e ben noti, è il modello più ragionevole, con rigorosi controlli per il passaggio da una fase all'altra

Modello a V (Ministero della difesa tedesco, 1992)



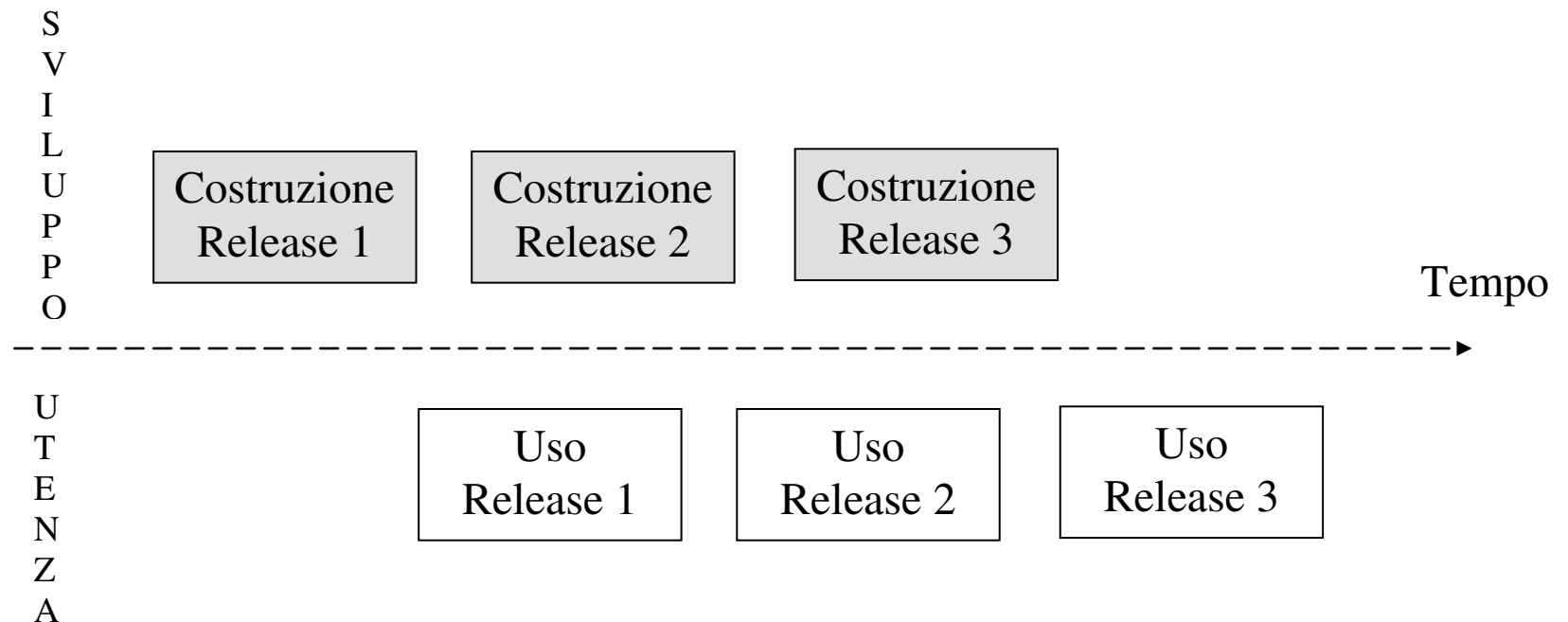
Modello a V (cont.)

- È una variante del modello a cascata che rende esplicita la necessità di effettuare iterazioni
- Mostra come le attività di testing siano collegate ad analisi dei requisiti e progetto
- I test di unità e di integrazione, oltre a occuparsi della correttezza dei programmi, possono essere usati per assicurarsi che tutti gli aspetti del progetto del programma siano stati implementati correttamente
- Il test di accettazione convalida l'aderenza ai requisiti associando un passo del test a ciascun elemento della specifica
- I problemi scoperti sul lato destro della V determinano la riesecuzione di attività sul lato sinistro

Modelli incrementali / iterativi (detti anche evolutivi)

Sono una risposta alla necessaria evoluzione del sw, alternativa interessante soprattutto quando i requisiti sono imperfetti o instabili: il sistema evolve man mano che i requisiti vengono progressivamente compresi

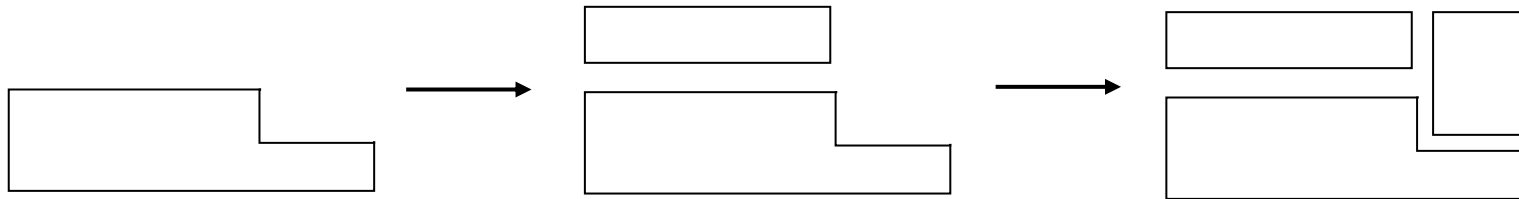
Sviluppo a stadi: mentre è operativa la release n del sistema, si lavora alla release $n+1$



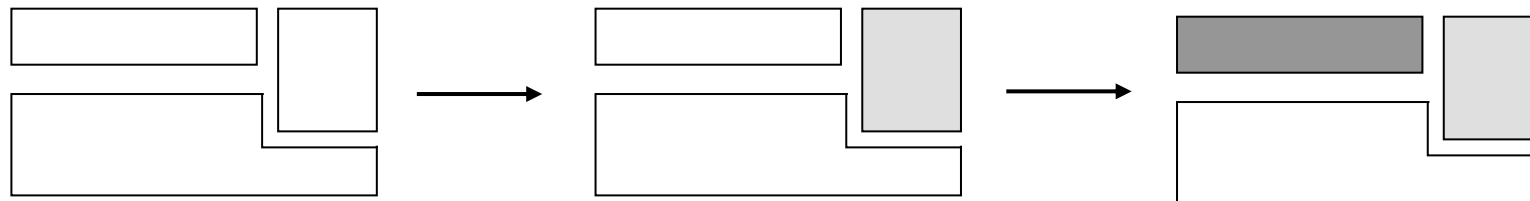
Modelli incrementali / iterativi (cont.)

Lo sviluppo a stadi è supportato da due approcci:

- sviluppo incrementale



- sviluppo iterativo

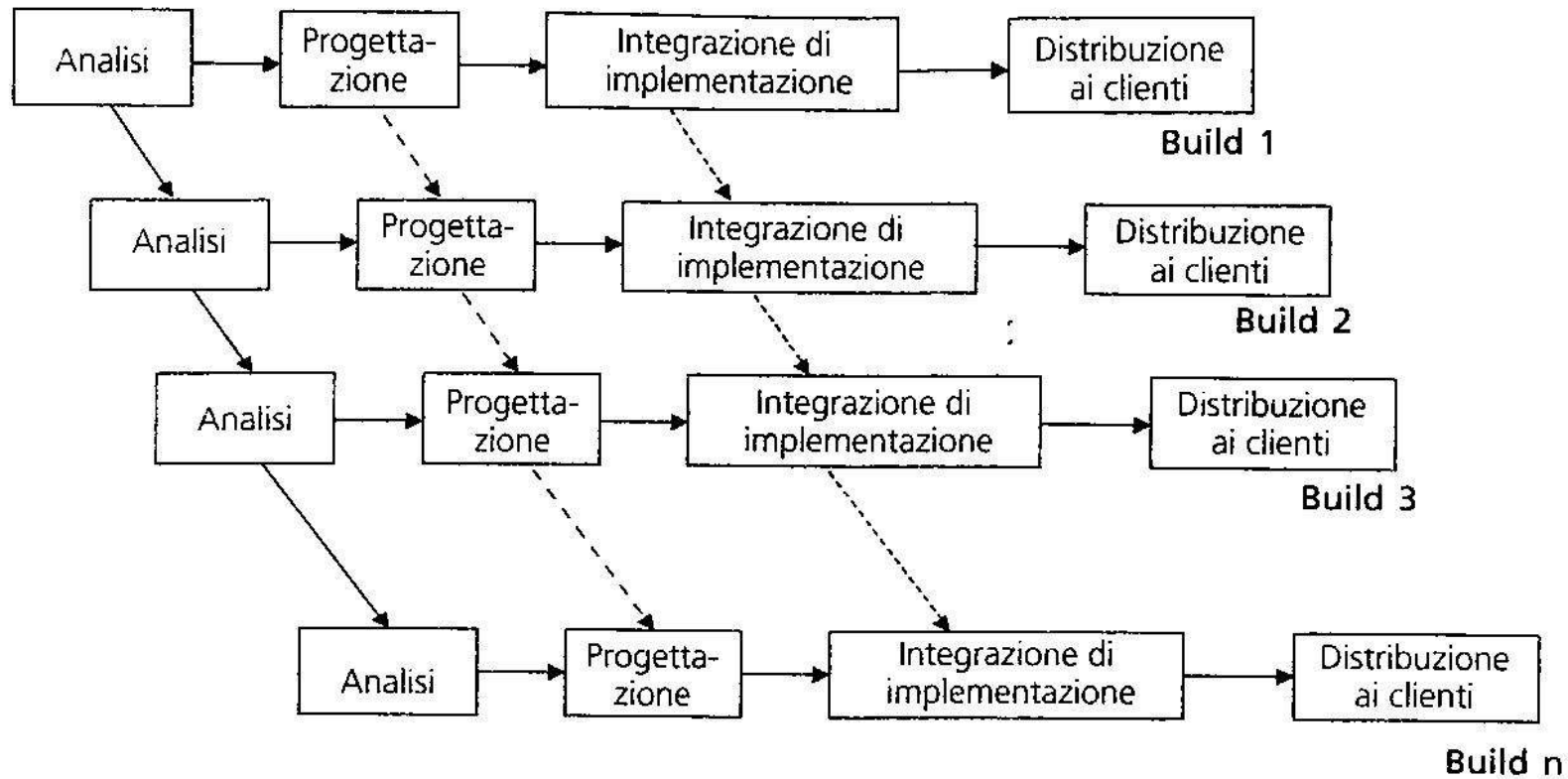


Modelli incrementali / iterativi (cont.)

- Le funzionalità individuate nell'analisi dei requisiti sono allocate a iterazioni diverse; ad ogni release segue la consegna di una nuova versione operativa con qualità e/o funzionalità aumentate
- Dall'incremento corrente si devono trarre indicazioni su come effettuare il successivo attraverso modifiche semplici e affidabili → necessità di metodi di progettazione opportuni
- Uso della prototipazione
Prototipo (= modello operativo dell'applicazione)
 - ✓ Usa e getta (ad es.
 - da mostrare al (potenziale) committente (prototipo dimostrativo)
 - per migliorare la comprensione da parte degli sviluppatori
 - per la convalida dei requisiti da parte di utenti/clienti)
 - ✓ Evolutivo (= primo incremento)

Conclusione: sono i modelli preferibili quando i rischi maggiori risiedono nell'instabilità e incertezza dei requisiti

Ingegneria concorrente



Modello a spirale (Boehm, 1988)

Determinazione
obiettivi, alternative
e vincoli

1

Valutazione alternative,
identificazione e
risoluzione rischi

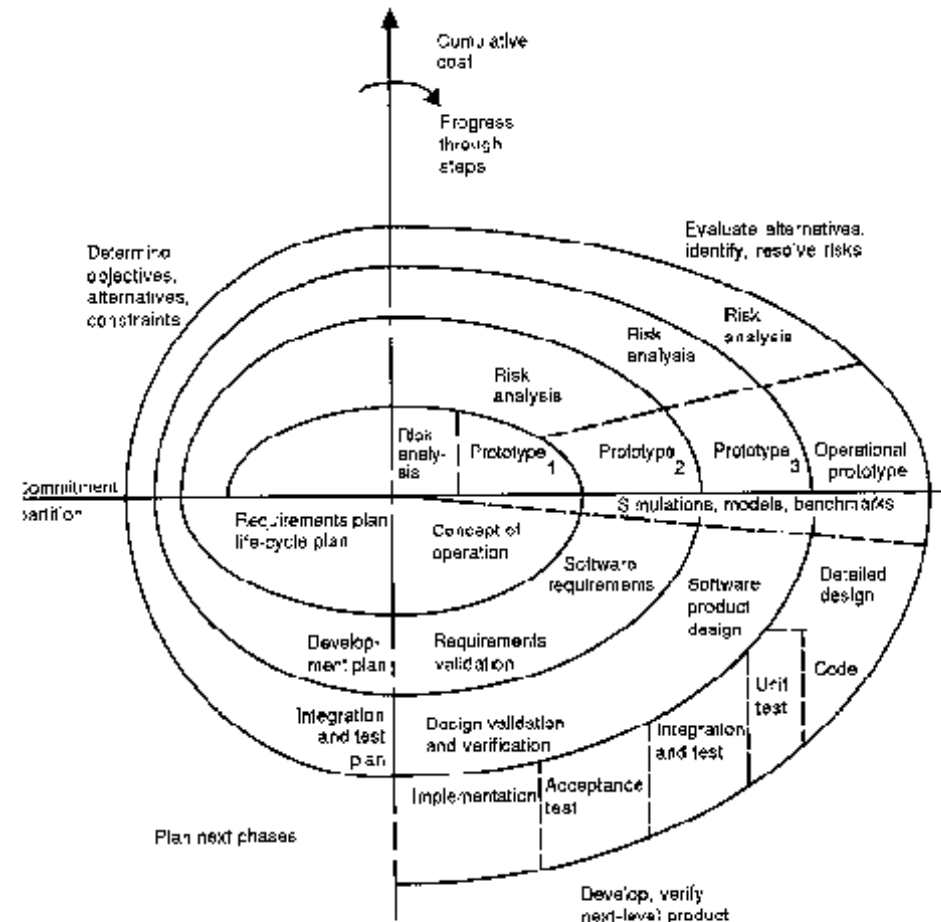
2

4

Pianificazione
fase successiva

3

Sviluppo e verifica
del prossimo livello
del prodotto



Modello a spirale (cont.)

- È un meta-modello dei processi sw
- Dati gli obiettivi di ciascuna iterazione, se ne valutano in dettaglio i rischi e si decide di conseguenza che alternativa adottare
- Quattro volute i cui prodotti sono (dalla più interna alla più esterna):
 - ✓ concept of operation (descrizione di alto livello di come il sistema dovrebbe funzionare)
 - ✓ requisiti
 - ✓ progetto
 - ✓ codifica e testing
- In ogni voluta è previsto l'uso di prototipi per valutare la fattibilità o desiderabilità di un'alternativa
- Raggio della spirale = costo accumulato nel progetto

Casi d'uso

- Introdotti in UML da Jacobson nel 1994 come elementi principali dello sviluppo del sw (ma il concetto era già stato pubblicato nel 1987)
- Sono un veicolo per la pianificazione di progetto (controllano lo sviluppo iterativo)
- Pilotano l'intero processo di sviluppo secondo il RUP
- La loro somma è l'immagine del sistema verso l'esterno

Caso d'uso: una definizione informale

Caso d'uso = fotografia di un particolare aspetto del sistema, tipica interazione fra un utente (anche automatico) e il sistema per ottenere un risultato, funzione che l'utente è in grado di capire e che ha valore per l'utente

Una buona fonte per identificare i casi d'uso sono gli eventi esterni a cui il sistema deve reagire

Caso d'uso: una definizione più precisa

Caso d'uso = insieme di *scenari* legati da un *obiettivo* comune dal punto di vista dell'utente

Obiettivo: gli obiettivi dell'utente sono requisiti del sistema

Scenario = sequenza di passi che descrivono l'interazione fra l'utente e il sistema

Esempio

Obiettivo: Acquisto di un prodotto su *www*

Scenari: carta di credito valida (scenario principale), carta di credito non valida (alternativa), cliente abituale (alternativa)

Un esempio di caso d'uso (gestione videoteca)

Nome	Aggiungi delegato
Attore	Addetto della videoteca
Scenario principale	<ol style="list-style-type: none">1. <<include>> “Cerca cliente” (per selezionare i dati del cliente che ha effettuato la richiesta di avere un ulteriore delegato)2. L’addetto sceglie la funzionalità “Nuovo delegato”3. Il sistema presenta i campi per l’aggiunta di un delegato4. L’addetto inserisce i dati in tali campi5. Il sistema chiede conferma6. L’addetto conferma7. Il sistema registra i dati del nuovo delegato e mostra la lista aggiornata dei delegati del cliente considerato <p>Postcondizione: il nuovo delegato è stato registrato (e pertanto può usare la tessera del cliente)</p> <p>Fine</p>
Scenario alternativo	<p>3a.Precondizione: il numero dei delegati del cliente considerato è già massimo</p> <p>Il sistema avverte che non si possono aggiungere nuovi delegati</p> <p>Fine</p>
Scenario alternativo	<p>6a.L’addetto non conferma</p> <p>6b.Il sistema consente all’addetto di modificare i dati del delegato già inseriti</p> <p>Torna al punto 5</p>

Casi d'uso e attori

Ogni caso d'uso

- ha necessariamente un attore principale, che è quello che richiede un servizio al sistema e, solitamente, dà inizio al caso
- può avere uno o più attori secondari, con i quali il sistema comunica nel tentativo di svolgere con successo il caso d'uso

Casi d'uso e requisiti

- Ogni caso d'uso può corrispondere a più requisiti funzionali
- Un requisito funzionale può dare origine a più casi d'uso
- A ogni caso d'uso possono venire associati più requisiti non funzionali

Casi d'uso e UML

- Il punto di vista da adottare nella descrizione di un caso d'uso è quello dell'utente che interagisce col sistema (attore), non quello del funzionamento interno del sistema
- Si solito si opera una descrizione testuale di ogni caso d'uso, mediante una sequenza completa di passi, ciascuno dei quali corrisponde a una interazione tra l'attore e il sistema
- Un caso può essere incluso in un altro semplicemente sottolineandone il nome, come per un collegamento ipertestuale
- UML non indica uno standard per descrivere testualmente i casi d'uso ma mette a disposizione i diagrammi dei casi d'uso per visualizzarli



Non è necessario usare i diagrammi UML dei casi d'uso
per utilizzare i casi d'uso

Casi d'uso: elementi del contenuto

La descrizione di un caso d'uso può comprendere:

- precondizioni = condizioni che devono essere verificate perché si possa dare inizio all'esecuzione del caso d'uso o condizioni che determinano il verificarsi di una sequenza di interazioni diversa da quella dello scenario principale
- postcondizioni (o garanzia o effetti) = ciò che il sistema assicura alla fine dello svolgimento di uno scenario
- trigger = evento che dà origine al caso d'uso

Casi d'uso: livello di dettaglio

- I casi d'uso non devono menzionare l'interfaccia utente (sono redatti prima della progettazione dell'interfaccia)
- Maggiore è il rischio connesso al caso d'uso, maggiore è il dettaglio richiesto nella sua descrizione
- Un buon livello di dettaglio facilita tutte le attività successive
- Troppi dettagli
 - ✓ Complicherebbero e allungherebbero inutilmente la descrizione
 - ✓ Introdurrebbero prematuramente scelte progettuali
- Nelle fasi successive a quella di elicitazione e analisi dei requisiti, si aggiungono ai casi d'uso i dettagli via via necessari all'implementazione (senza necessariamente scriverli)

Casi d'uso: individuazione

Si ricavano dalle interviste con committenti e utenti finali del sistema mediante un processo di definizione iterativo:

- presumibilmente si inizia identificando i comportamenti più semplici
- si descrivono i comportamenti alternativi e più complessi

Nel RUP è bene identificare i casi d'uso e svolgere la modellazione concettuale del dominio contemporaneamente, sempre insieme agli utenti

Passi (non rigidi) per l'individuazione dei casi d'uso

1. Definizione lista attori
2. Identificazione degli obiettivi di ogni attore
3. Per ogni coppia (attore, obiettivo) descrizione delle interazioni (desiderate) col sistema
4. A ogni passo di uno scenario principale, individuazione delle condizioni di estensione
5. Redazione delle estensioni (scenari alternativi)

Casi d'uso: impiego

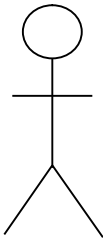
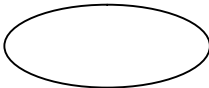
- Elicitazione e analisi dei requisiti
- Organizzazione del progetto
- Generazione dei casi di test

Casi d'uso: livelli


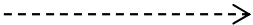
Classificazione secondo Cockburn (2001) <http://usecases.org>

- Sea-level: casi d'uso che rappresentano un'interazione tra un attore e il sistema
- Fish-level: casi d'uso che esistono solo perché inclusi in quelli sea-level
- Kite-level: casi d'uso che mostrano il ruolo dei casi d'uso sea-level all'interno di interazioni di business più ampie

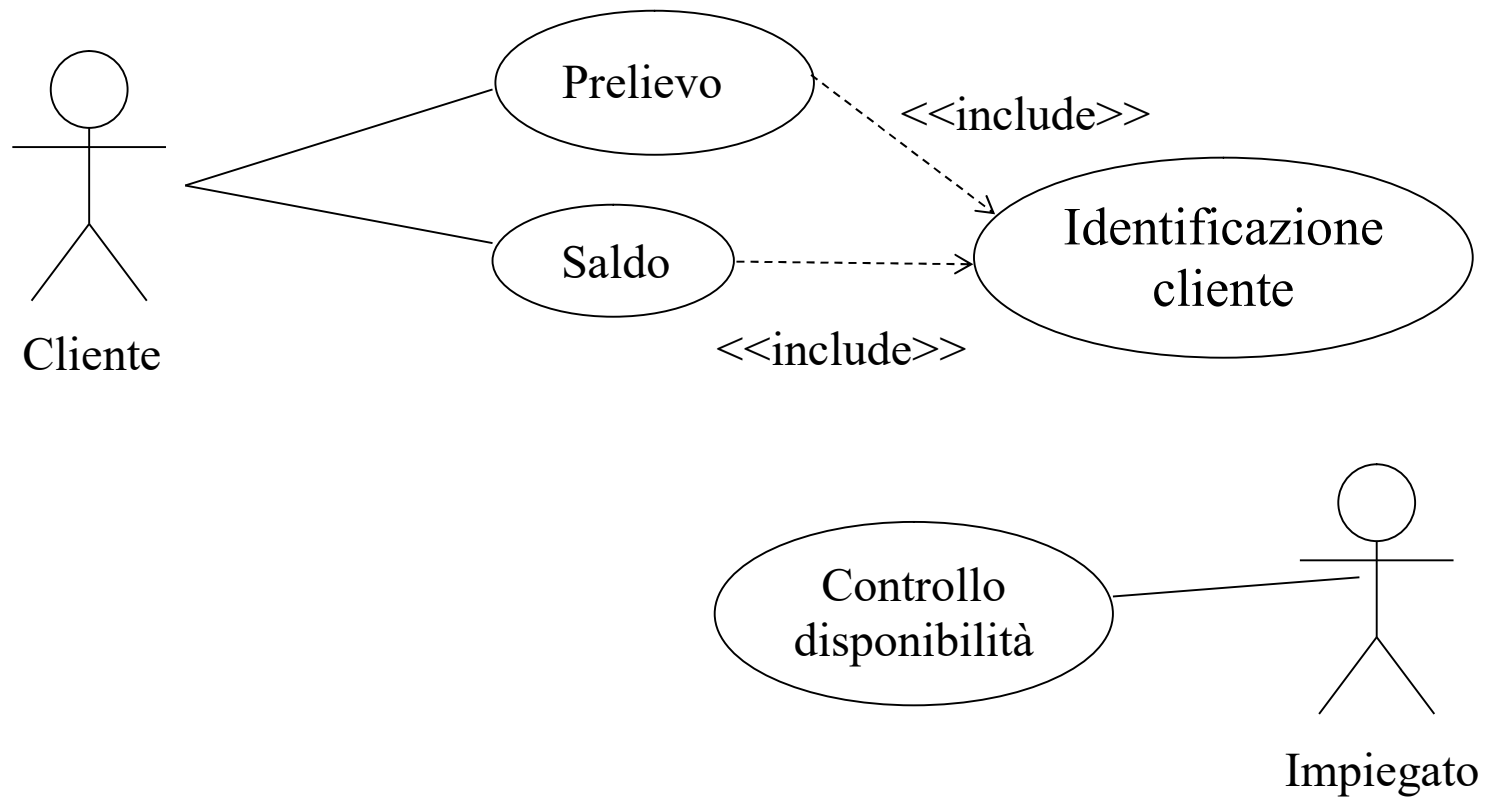
Diagrammi dei casi d'uso

Elementi	Sintassi	Semantica
Attore (il termine più appropriato sarebbe <i>ruolo</i>)	Omino stilizzato + nome 	<ul style="list-style-type: none"> • Ruolo interpretato da una categoria di utenti (esseri umani, organizzazioni, enti, istituzioni, computer o sistemi esterni) nei confronti del sistema • La stessa categoria di utenti può anche interpretare più ruoli distinti, oppure gli stessi utenti possono far parte di più categorie
Caso d'uso	Ellisse contenente il nome del caso d'uso 	

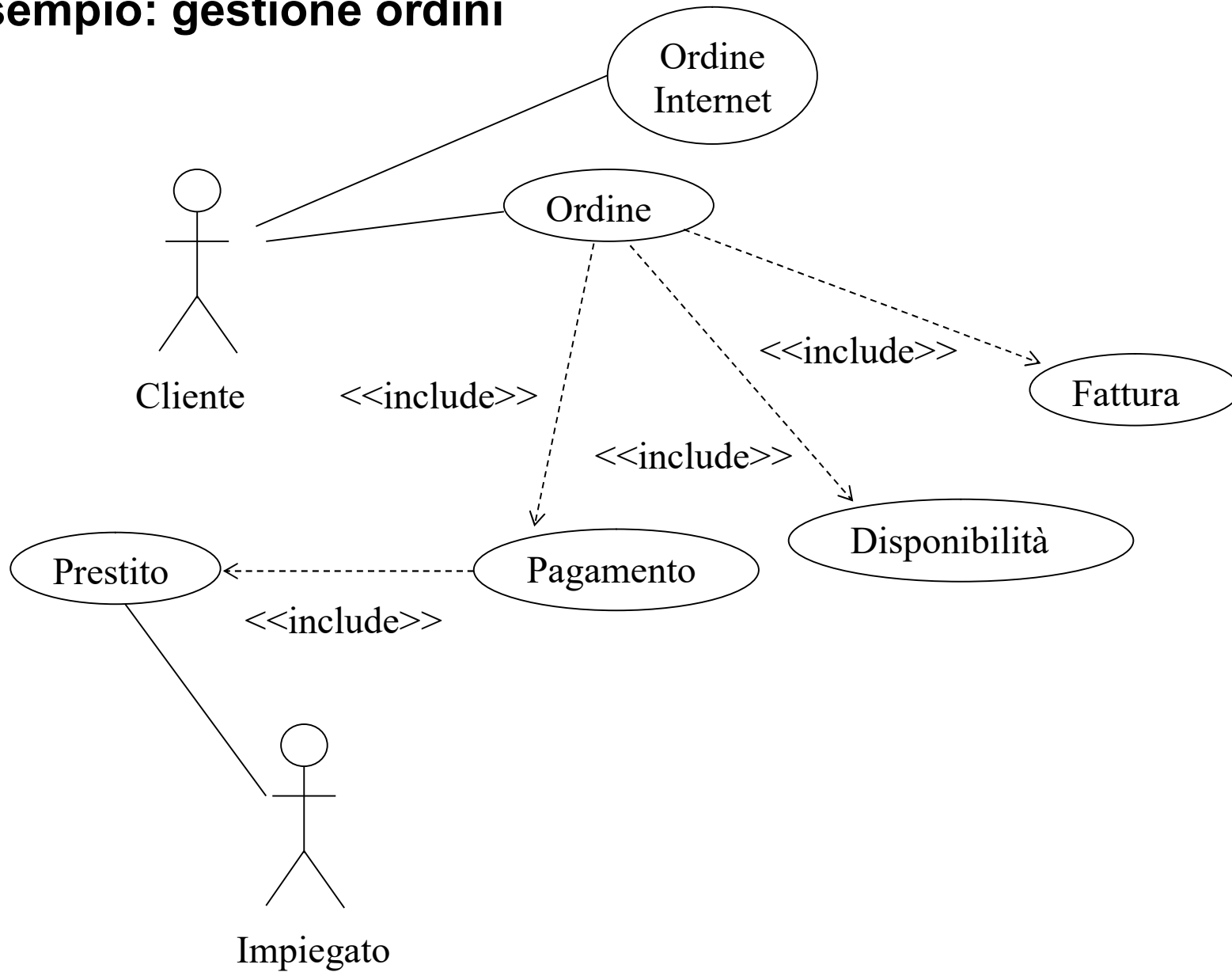
Diagrammi dei casi d'uso (cont.)

Elementi	Sintassi	Semantica
Collegamento fra attore e caso d'uso	<p>Linea continua </p> <p>Un singolo attore può essere collegato a più casi d'uso</p> <p>Un caso d'uso può essere collegato a più attori</p>	<ul style="list-style-type: none"> • L'attore esegue il caso d'uso collegato, ottenendo valore da esso • L'attore può essere anche passivo (es. il sistema deve inviare al cliente la fattura per l'acquisto effettuato, senza che il cliente l'abbia richiesta)
Relazione di inclusione fra casi d'uso	<p>Freccia a una punta, con linea tratteggiata, uscente dal caso d'uso includente e terminante nel caso d'uso incluso +</p> <p><i><<include>></i></p> <p></p>	Un comportamento si ripete in più casi d'uso e non si vuole ripetere la sua descrizione (testuale)

Esempio: Bancomat



Esempio: gestione ordini



Unified Modeling Language (UML)

- È una famiglia di notazioni grafiche che si basano su un singolo meta-modello
- Serve per definire, progettare, realizzare e documentare sistemi sw (in particolare quelli OO)
- Copre l'intero ciclo di vita del sw senza imporre alcun processo di sviluppo predefinito
- È indipendente da qualsiasi linguaggio di programmazione
- È utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni
- È basato sui modelli, che sono uno strumento per gestire la complessità
- È estensibile (per modellare meglio le diverse realtà)
- È sponsorizzato dalle maggiori case produttrici di sw



UML: cronologia

Nato dalla fusione di molti linguaggi grafici di modellazione OO sviluppati tra la fine degli anni '80 e l'inizio dei '90. **Le versioni UML 0.9 e 0.91, del 1996, sono opera di Grady Booch, Jim Rumbaugh e Ivar Jacobson (detti “los tres amigos”)**

Novembre 1997: UML 1.1 (risultato della fusione di UML 1.0, rilasciata da Rational Software, con altre proposte) diventa uno standard OMG

Inizio 1999: UML 1.3

Giugno 2003: UML 2.0 (diventa uno standard OMG nel 2005; le versioni 1.4.2.e 2.4.1 sono anche standard ISO/IEC)

Rational Software **dal 2003** fa parte di IBM

A partire dalla versione 2.0, il nucleo (“core”) di UML è restato sostanzialmente stabile. Nelle versioni successive è proseguita invece l'evoluzione dei profili UML, ossia l'estensione di UML per supportare nuovi ambiti di applicazione e nuove tecnologie

Versioni standardizzate di UML

HISTORY

FORMAL VERSIONS

VERSION	ADOPTION DATE	URL
2.5.1	December 2017	https://www.omg.org/spec/UML/2.5.1
2.4.1	July 2011	https://www.omg.org/spec/UML/2.4.1
2.3	May 2010	https://www.omg.org/spec/UML/2.3
2.2	January 2009	https://www.omg.org/spec/UML/2.2
2.1.2	October 2007	https://www.omg.org/spec/UML/2.1.2
2.0	July 2005	https://www.omg.org/spec/UML/2.0
1.5	March 2003	https://www.omg.org/spec/UML/1.5
1.4	September 2001	https://www.omg.org/spec/UML/1.4
1.3	February 2000	https://www.omg.org/spec/UML/1.3
1.2	July 1999	https://www.omg.org/spec/UML/1.2
1.1	December 1997	https://www.omg.org/spec/UML/1.1

796 pagine di
specifiche



Profili UML

NAME	ACRONYM	VERSION	STATUS	PUBLICATION DATE
UML Profile for BPMN Processes	BPMNProfile™	1.0	formal	luglio 2014
UML Profile for CORBA and CORBA Components	CCCMP™	1.0	formal	aprile 2008
UML Profile for CORBA Components	CCMP™	1.0	formal	luglio 2005
UML Profile for CORBA	CORP	1.0	formal	aprile 2002
UML Profile for Enterprise Application Integration	EAI™	1.0	formal	marzo 2004
UML Profile for Enterprise Distributed Object Computing	EDOC™	1.0	formal	febbraio 2004
UML Profile for MARTE	MARTE	1.2	formal	aprile 2019
UML Profile for NIEM	NIEM-UML™	3.0	formal	aprile 2017
UML Profile for Modeling QoS and FT	QFTP	1.1	formal	aprile 2008
Software Radio Components	SDRP™	1.0	formal	marzo 2007
UML Profile for System on a Chip	SoCP™	1.0.1	formal	agosto 2006
UML Profile for Schedulability, Performance, & Time	SPTP™	1.1	formal	gennaio 2005
UML Profile for Telecommunication Services	TelcoML™	1.0	formal	luglio 2013
SES Management TelcoML Extension	TelcoML-SEST™	1.0	formal	agosto 2013
UML Profile for ROSETTA	UPR	1.0 beta	beta	luglio 2018
UML Testing Profile 2	UTP2	2.0	formal	dicembre 2018
UML Profile for Voice-based Applications	VOICP™	1.0	formal	aprile 2008
Total: 17				

Object Management Group (OMG)

<http://www.omg.org>



- È un'organizzazione (consorzio) **fondata nel 1989** a cui aderiscono **centinaia di** aziende (leader in campo internazionale)
- Slogan: “Setting vendor-neutral software standards, and enabling distributed, enterprise-wide interoperability”
- Obiettivo: produrre e mantenere un corpo di specifiche che supportino tutte le fasi del ciclo di vita di sw distribuito ed eterogeneo
- Le specifiche sono scritte, influenzate e adottate dalle aziende aderenti
- Chiunque può scaricare gratuitamente le specifiche dal sito web del gruppo
- Ogni azienda, istituzione, organizzazione pubblica può divenire membro del gruppo
- Specifiche OMG:
 - ✓UML (standardizza le rappresentazioni di analisi e progettazione)
 - ✓CORBA (Common Object Request Broker Architecture, fissa gli standard per l'interoperabilità delle applicazioni)

Diagrammi UML

Diagrammi strutturali

- delle classi
- dei componenti
- di struttura composita
- di deployment
- degli oggetti
- dei package

Diagrammi comportamentali

- di attività
 - dei casi d'uso
 - di stato (o di macchina a stati)
 - di sequenza
 - di comunicazione
(ex collaborazione)
 - di interazione generale
 - di temporizzazione
- } Diagrammi di interazione

Spesso si possono usare elementi di un tipo di diagramma all'interno di un altro

Nessuno comprende o utilizza UML nella sua interezza; la maggior parte delle persone ne usa un piccolo sottoinsieme

Modalità di utilizzo di UML

- Abbozzo (sketch) – su cui si concentra il testo di Fowler
- Progetto dettagliato (blueprint)
- Linguaggio di programmazione

UML come abbozzo

Diagrammi informali (schizzi, bozzetti) incompleti (cioè concentrati solo sulle informazioni più importanti) di parti selezionate del sistema, realizzati, tipicamente su lavagna, in poco tempo e in modo collaborativo dai programmatori in fase di

- Forward engineering (cioè prima di scrivere il codice), al fine di
 - Aiutare la comunicazione e discussione delle idee
 - Considerare alternative
 - Pianificare giorni/settimane di programmazione
- Reverse engineering (cioè a sistema esistente, quando si intende costruire un diagramma UML dello stesso per comprenderlo meglio), al fine di
 - Spiegare come funziona una parte del sistema

Per la loro espressività possono essere inclusi anche nella documentazione del SW

Sono abbozzi (per la loro incompletezza) la maggior parte dei diagrammi UML inclusi nei libri

UML come progetto

Modelli completi dettagliati di un sistema/sottosistema, realizzati, tipicamente usando strumenti CASE, in fase di

- Forward engineering dal progettista e forniti ai programmatori per documentare senza ambiguità tutte le decisioni di progetto da implementare (spesso con lo scopo di ridurre la programmazione a un'attività semplice e relativamente meccanica); Fowler crede che siano difficili da realizzare e rallentino il processo di sviluppo
- Reverse engineering al fine di documentare il codice esistente

UML come linguaggio di programmazione

Diagrammi (di interazione, di stato e di attività) che vengono compilati direttamente in codice eseguibile → codice sorgente e rappresentazione UML coincidono e la nozione di forward e reverse engineering non ha più senso

Significato dei diagrammi

Due prospettive

- Software
 - Di specifica (interfaccia)
 - Di implementazione (es. conversione di codice sorgente in diagrammi UML)
- Concettuale (cioè relativa al dominio applicativo), che ha lo scopo di costruire un vocabolario comune (es. uso di diagrammi UML per comprendere i molti significati dell'espressione *beni patrimoniali* con un gruppo di contabili)

Le prospettive non appartengono a UML, però il significato di ogni elemento di un diagramma dipende dalla prospettiva adottata

Notazione e meta-modello

Definizione di UML = notazione + meta-modello

Notazione = sintassi grafica del linguaggio di modellazione = insieme degli elementi grafici di ciascun diagramma, dove ogni elemento grafico rappresenta un concetto



quesito: qual è il significato di ciascun concetto?

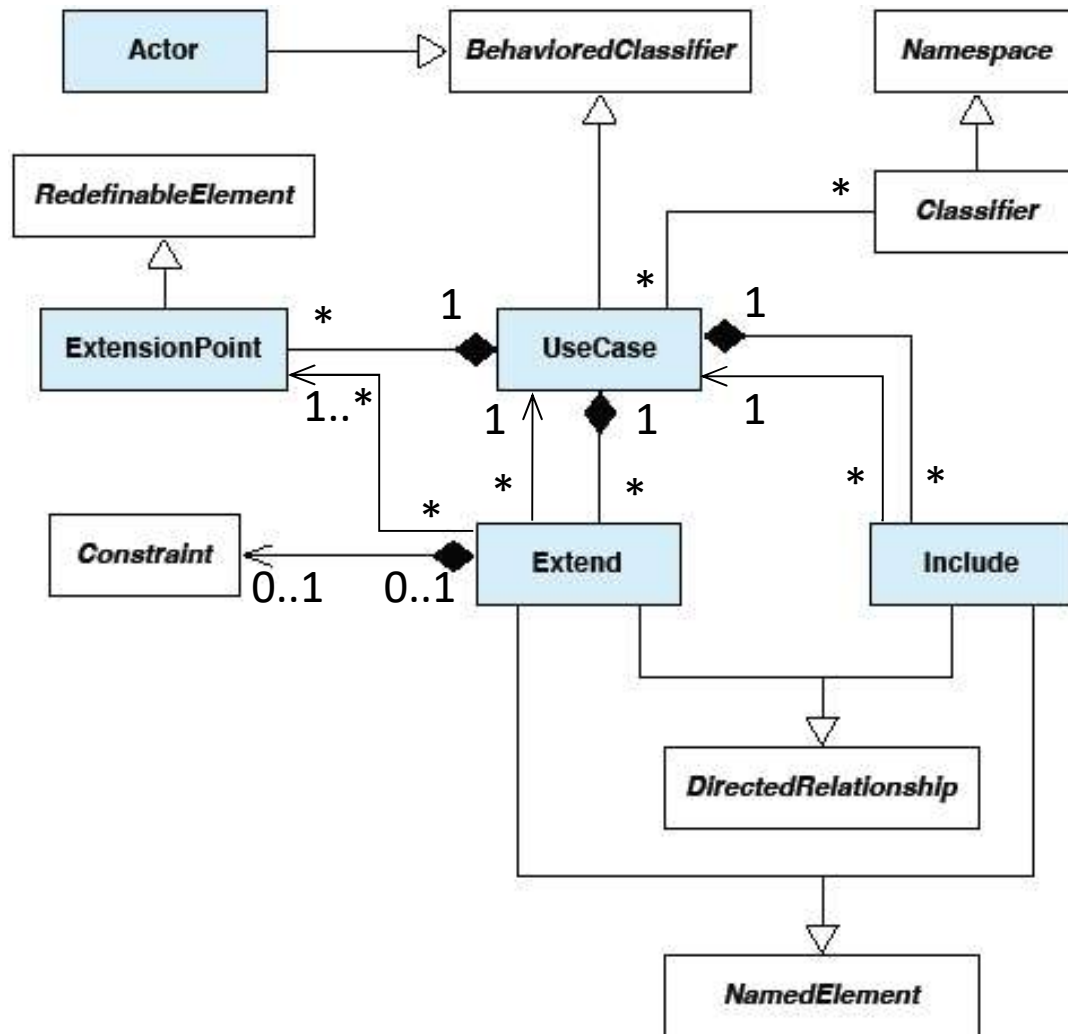
risposta: manca una definizione formale



Meta-modello = diagramma (solitamente diagramma delle classi) che definisce i concetti del linguaggio (è importante se si vuole usare UML come linguaggio di programmazione)

∄ una definizione formale di corrispondenza fra UML e un qualsiasi linguaggio di programmazione

Metamodello (semplificato) del diagramma dei casi d'uso



Quale UML è legale?

UML nella pratica odierna è definito

- sia da regole prescrittive, dettate dallo standard OMG → uso standard (o legale o normativo o ben formato), l'unico consentito come linguaggio di programmazione
- sia da regole descrittive, il cui significato è stabilito per convenzione comune → uso convenzionale

Ulteriore complessità deriva

- da UML 2, che stabilisce notazioni contrastanti con quelle date in UML 1 o con l'uso comune
- dal principio UML di “soppressione”, secondo cui qualsiasi informazione può essere soppressa → quando si interpreta un diagramma, di fronte a un'info mancante non si sa se l'autore intendesse semplicemente ometterla o invece usare il valore di default (incluso nel meta-modello)

UML: Motivazioni

Si usa la notazione semiformale UML perché:

- favorisce la comunicazione fra i membri del gruppo di sviluppo
- favorisce la comunicazione con i clienti (in particolare, grazie ai casi d'uso)
- è di grande aiuto nella fase di elicitazione e analisi dei requisiti (collocata entro la fase di Elaborazione del RUP)
- aiuta a sfruttare i vantaggi dell'OO (i linguaggi OO permettono tali vantaggi ma non li forniscono)
- esistono strumenti CASE basati su UML

Diagrammi di interazione

- Diagrammi di sequenza
- Diagrammi di comunicazione (ex collaborazione)
- Diagrammi di interazione generale
- Diagrammi di temporizzazione

Descrivono la collaborazione di un gruppo di oggetti per implementare collettivamente un comportamento

Diagrammi di sequenza

Ogni diagramma

- illustra tipicamente il comportamento dinamico del sistema in corrispondenza (di un singolo scenario) di un singolo caso d'uso
- descrive in un'unica vista il modo in cui gruppi di oggetti possono collaborare, scambiandosi messaggi (cioè invocando operazioni), nello svolgimento del caso d'uso
- enfatizza l'ordinamento temporale delle chiamate di metodi
- rende esplicita la struttura dei messaggi → evidenzia l'eventuale eccessiva centralizzazione di progetti dove un solo oggetto compie tutto il lavoro
- non è adeguato per la rappresentazione della logica di controllo (mostra dei limiti quando si devono rappresentare processi con molti cicli o condizioni)
- non definisce con precisione il comportamento degli oggetti, cioè non spiega i dettagli degli algoritmi
- presenta limitazioni nel rappresentare il comportamento polimorfo

Diagramma di sequenza



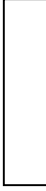
Elementi	Sintassi	Semantica
Partecipante (termine che però non fa parte di UML)	Scatola, posta in cima a una linea di vita, contenente <i>nome : classe</i> dove sia <i>nome</i> , sia <i>: classe</i> sono opzionali ma non possono mancare entrambi 	Solitamente (ma non necessariamente) indica un'istanza di una classe. Di seguito <u>assumiamo si tratti sempre di un oggetto</u>
Linea di vita	Linea tratteggiata verticale Il verso del tempo è dall'alto verso il basso 	Vita dell'oggetto (formalismo introdotto da Jacobson)
Barra di attivazione	Elemento opzionale: sottile scatola verticale sovrapposta alla linea di vita di un oggetto 	Intervallo in cui l'oggetto è attivo nell'interazione (ovvero un suo metodo è in esecuzione)

Diagramma di sequenza

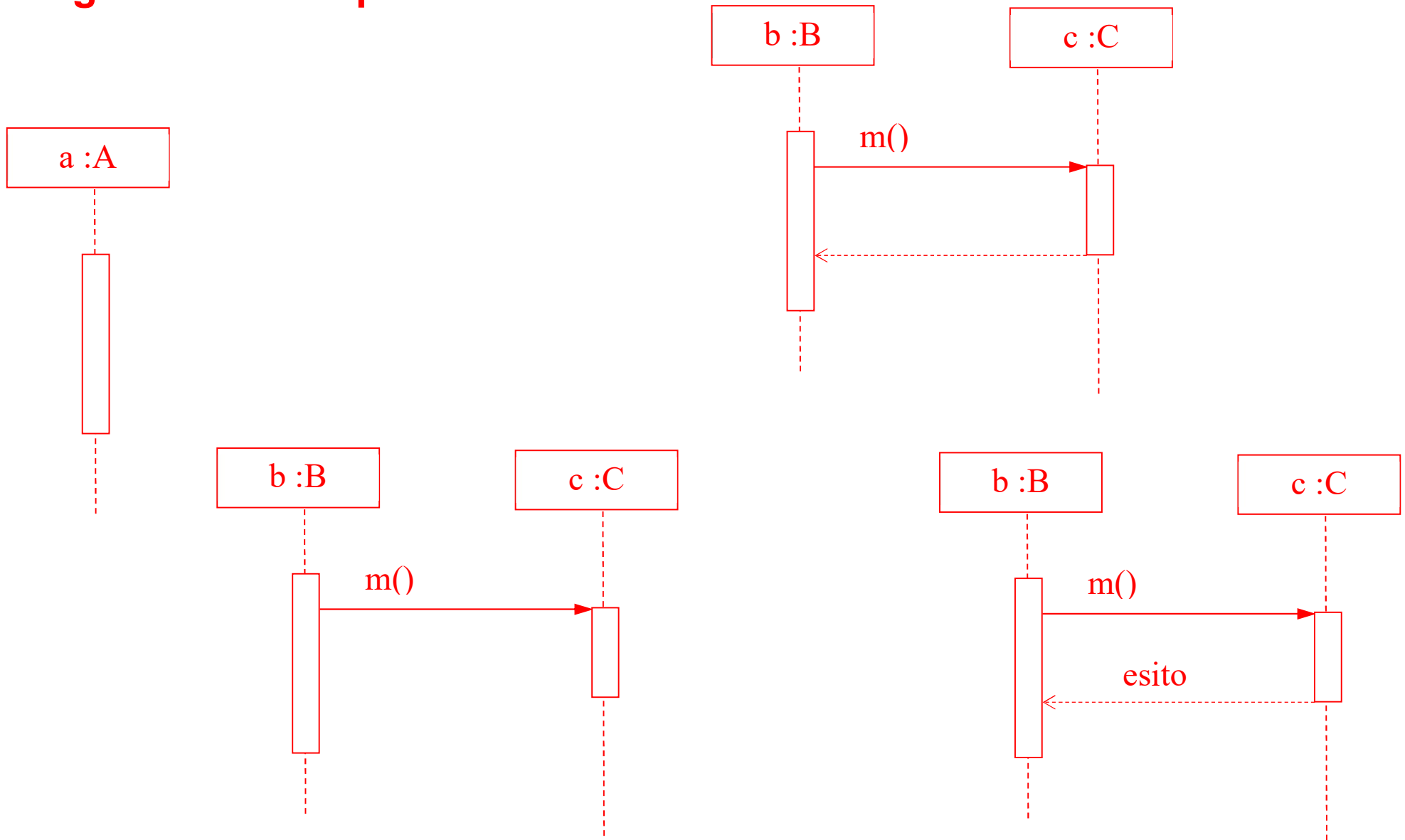


Diagramma di sequenza (cont.)

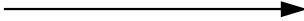
Elementi	Sintassi	Semantica
Messaggio di invocazione	<p>Freccia con punta piena e linea continua, posta tra due linee di vita (non necessariamente distinte) ed etichettata col nome del messaggio</p>  <p>Il nome del messaggio può essere seguito da un elenco di parametri fra parentesi tonde, dove ogni parametro può essere accompagnato dal suo tipo (introdotto da :) ed è separato dal successivo mediante una virgola</p> <p>L'ordine temporale di scambio dei messaggi è dall'alto verso il basso</p>	<ul style="list-style-type: none"> • Invocazione da parte dell'oggetto dalla cui linea di vita si diparte la freccia di un metodo dell'oggetto verso la cui linea di vita la freccia è diretta • Se le linee di vita sorgente e destinazione coincidono, si tratta di una <i>chiamata interna</i>

Diagramma di sequenza

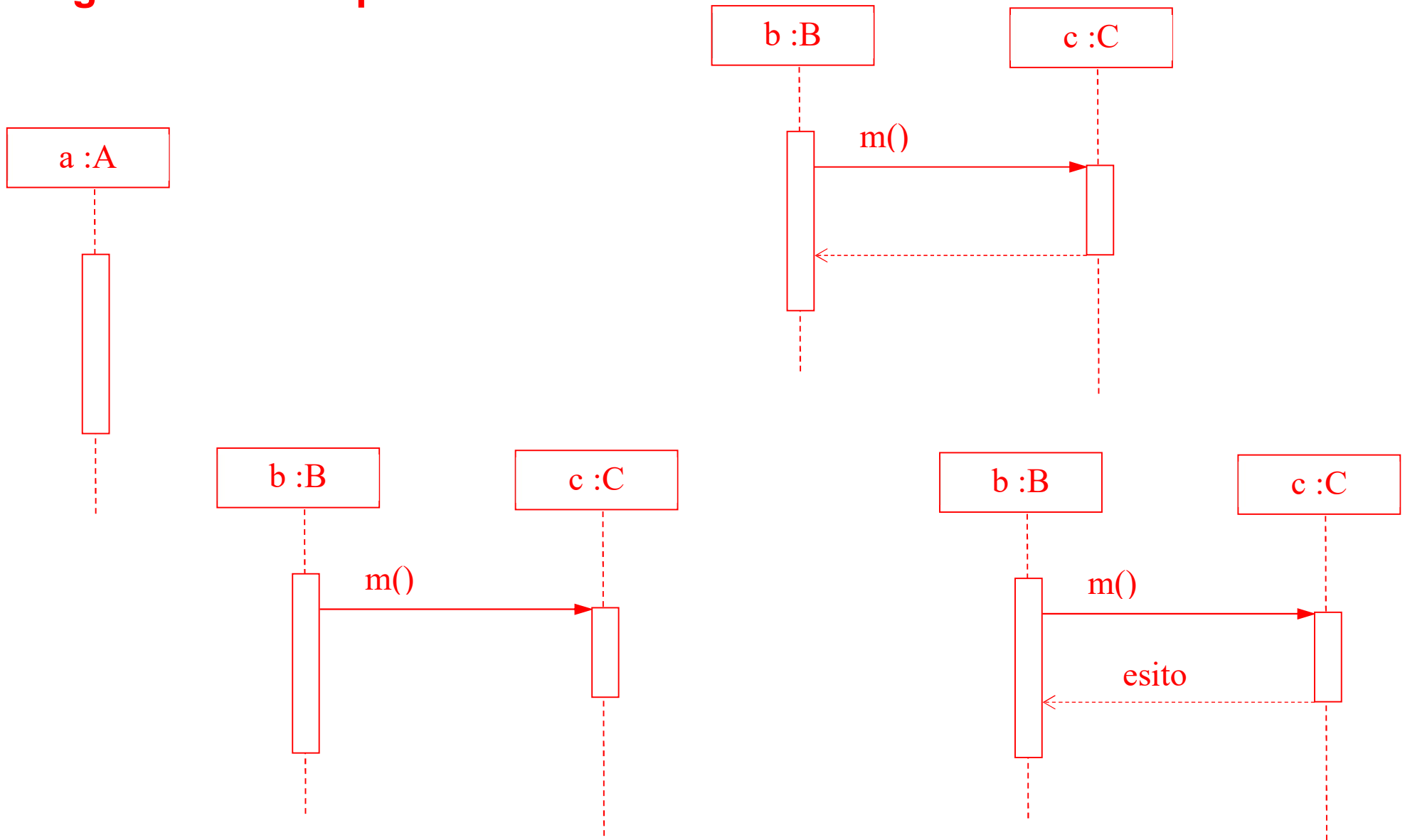


Diagramma di sequenza (cont.)

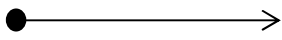
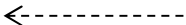
Elementi	Sintassi	Semantica
Messaggio trovato (<i>found message</i>)	Come il messaggio di invocazione ma l'origine della freccia è una pallina nera piena 	Messaggio che proviene da una fonte esterna e innesca l'interazione visualizzata
Messaggio di ritorno	Elemento opzionale: freccia con punta aperta e linea tratteggiata, posta tra due linee di vita distinte: la linea destinazione aveva precedentemente inviato un msg alla linea sorgente  Può essere accompagnato da etichetta riportante il nome del parametro di ritorno e/o il suo valore	Indica il valore ritornato da un'invocazione L'uso è consigliato solo se aggiunge effettivamente info

Diagramma di sequenza

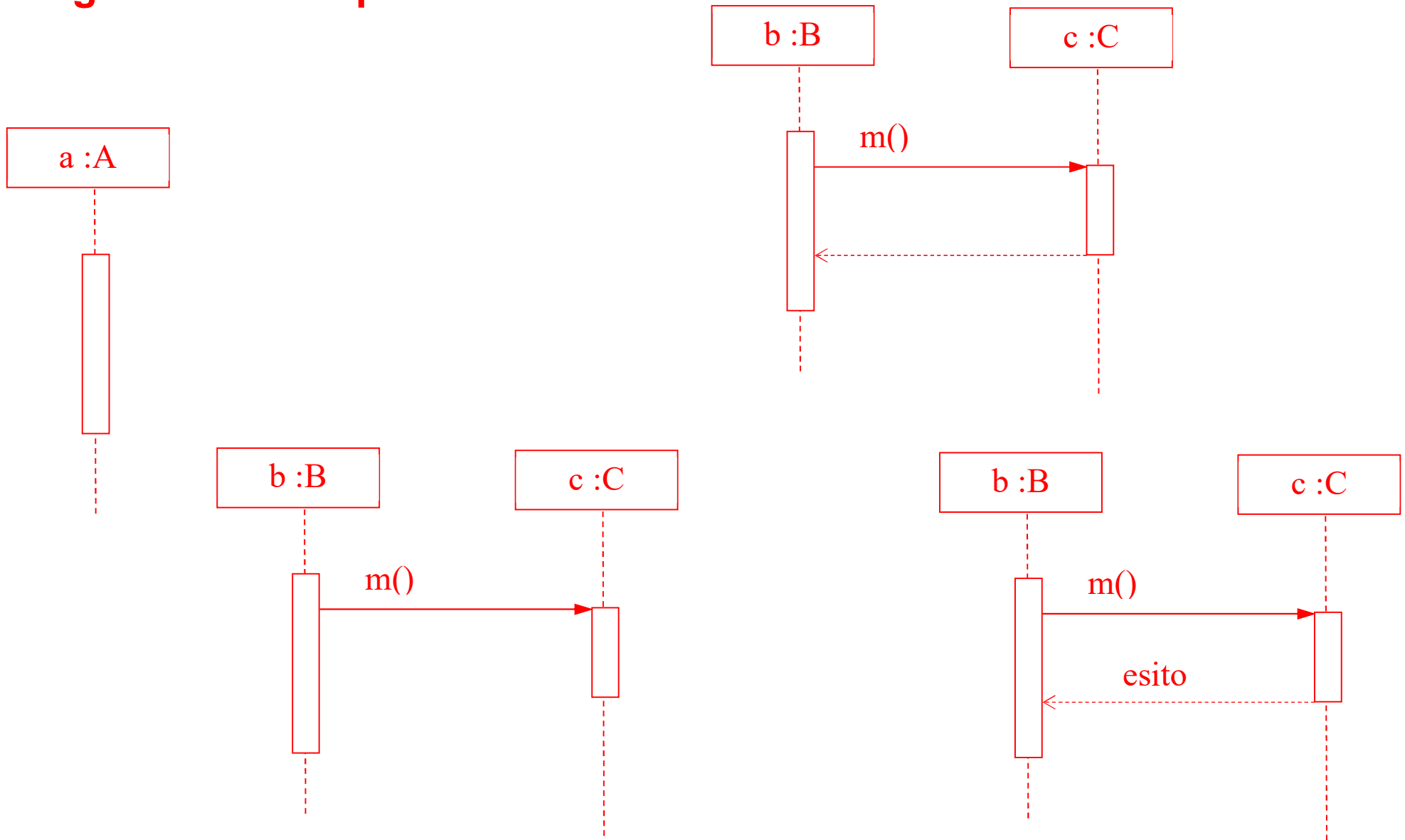


Diagramma di sequenza (cont.)

Elementi	Sintassi	Semantica
Messaggio di creazione di un oggetto	<p>È come un messaggio di invocazione ma termina contro la scatola che rappresenta l'oggetto</p> <p>Se il nuovo oggetto fa qualcosa immediatamente dopo la creazione, la sua barra di attivazione può essere direttamente attaccata alla scatola dell'oggetto stesso</p>	<p>Creazione di un'istanza di una classe</p> <p>Si può etichettare la freccia con la stringa <i>new</i> (che però non fa parte di UML)</p>

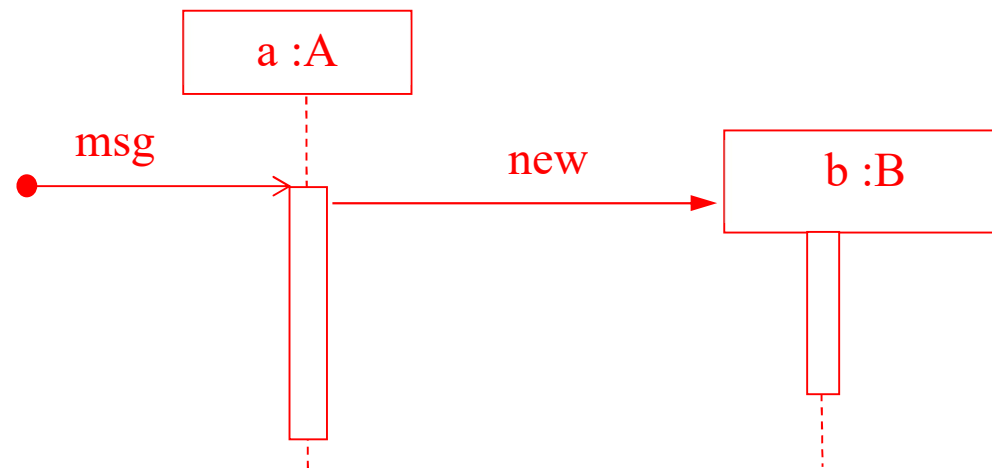


Diagramma di sequenza (cont.)

Elementi	Sintassi	Semantica
Messaggio di distruzione di un oggetto	È come un messaggio di invocazione ma termina in una grande X posta sotto la barra di attivazione dell'oggetto che viene distrutto	Distruzione dell'istanza di un oggetto In un ambiente dotato di garbage collection gli oggetti non sono distrutti esplicitamente ma è bene usare la X per indicare quando un oggetto non serve più ed è pronto per essere cancellato automaticamente
Autodistruzione di un oggetto	Grande X posta sotto la barra di attivazione dell'oggetto	

Esempio

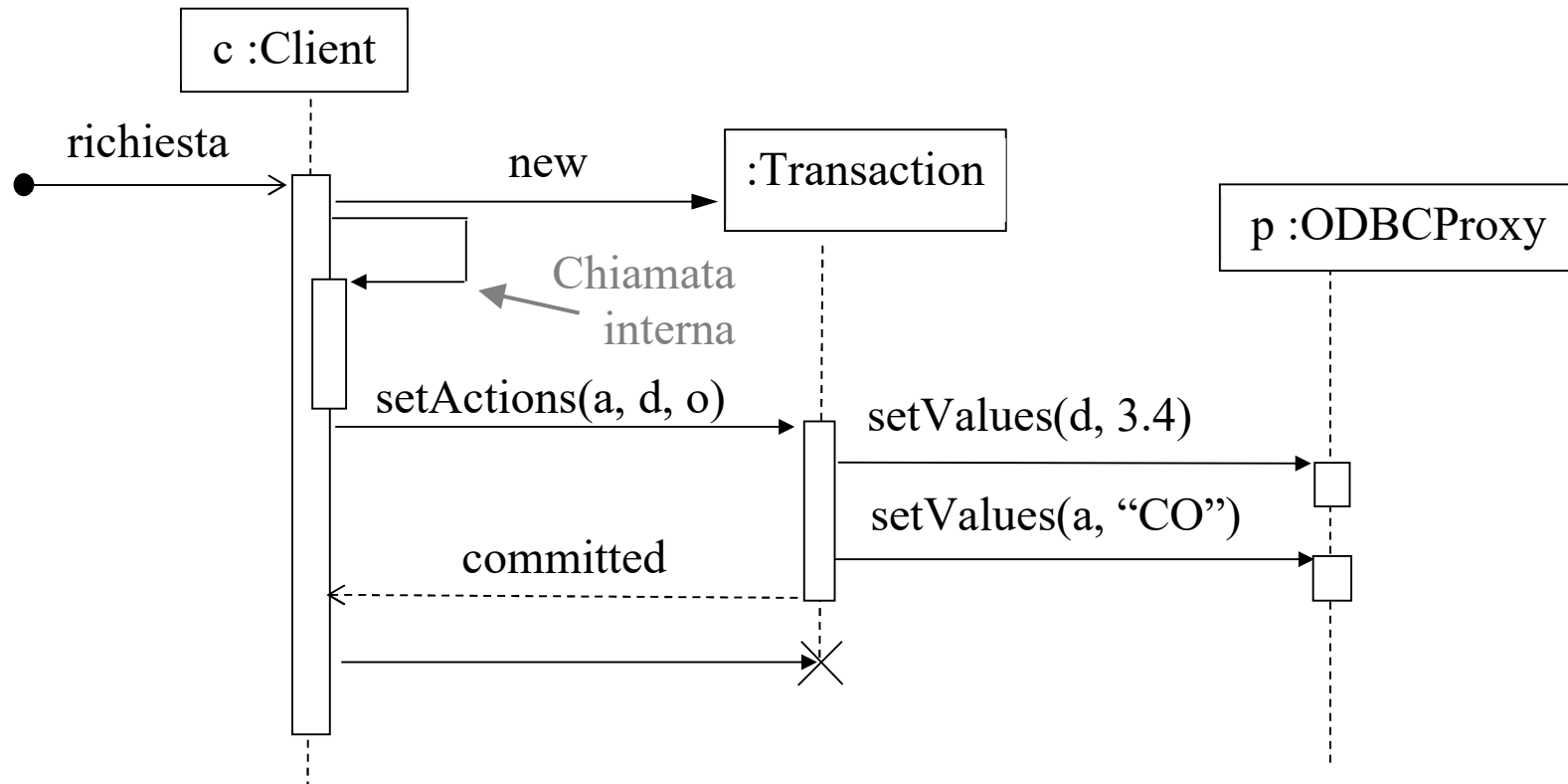


Diagramma di sequenza (cont.)

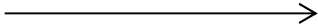

Elementi	Sintassi	Semantica
Messaggio asincrono	Freccia in linea continua a punta aperta 	Messaggio che non blocca l'esecuzione del chiamante, il quale può proseguire con la sua elaborazione Pratica comune nelle applicazioni multithread, dove serve, ad es. per creare un thread o per comunicare con un thread già in esecuzione

Diagramma di sequenza (cont.)

Elementi	Sintassi	Semantica
Frame di interazione 	Riquadro, dotato di operatore, che seleziona uno o più frammenti del diagramma, ciascuno dei quali può essere dotato di guardia L'operatore è scritto nell'angolo superiore sx del frame	Novità di UML 2 Serve per rappresentare la logica di controllo
Guardia	Espressione condizionale contenuta fra parentesi quadre	Il messaggio/ frammento associato viene inviato/ eseguito solamente se la condizione è verificata

Alcuni operatori

Operatore	Sintassi	Semantica
alt	Il frame evidenzia più frammenti contenuti in <i>tracce</i> , cioè reciprocamente separati mediante una linea orizzontale tratteggiata, ciascuno dotato di guardia che esprime una condizione mutuamente esclusiva rispetto a quella degli altri frammenti	Rappresenta una alternativa: viene eseguito solo il frammento la cui condizione è vera
opt	Il frame evidenzia un singolo frammento dotato di guardia (equivale ad alt con una sola traccia)	Il frammento viene eseguito solo se la condizione è vera
par	Il frame evidenzia più frammenti contenuti in tracce	Ogni frammento è eseguito in parallelo

Alcuni operatori (cont.)

Operatore	Sintassi	Semantica
region	Il frame evidenzia un singolo frammento privo di guardia	Rappresenta una regione critica: il frammento è eseguibile da un solo thread per volta
loop	Il frame evidenzia un singolo frammento dotato di guardia	Il frammento può essere eseguito più volte; la base dell'iterazione è indicata dalla guardia
neg	Il frame evidenzia un singolo frammento privo di guardia	Il frammento mostra una interazione non valida
ref	Il frame <ul style="list-style-type: none">• ricopre le linee di vita coinvolte nell'interazione a cui si riferisce,• indica il nome di tale interazione ed eventuali parametri,• può avere un valore di ritorno	Il frame si riferisce a un'interazione descritta da un altro diagramma (solitamente racchiuso in un frame con operatore sd)

Esempio con operatori

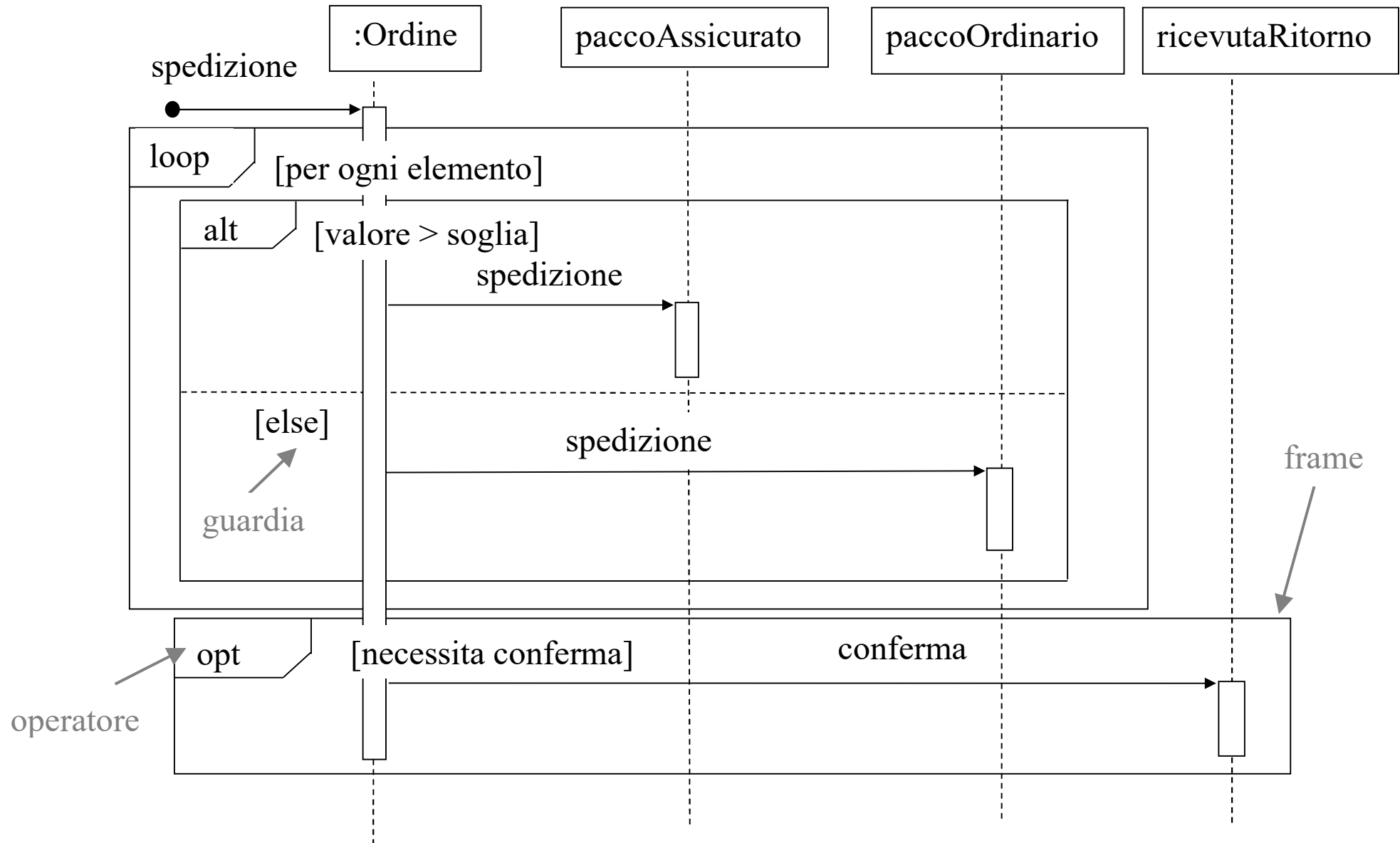
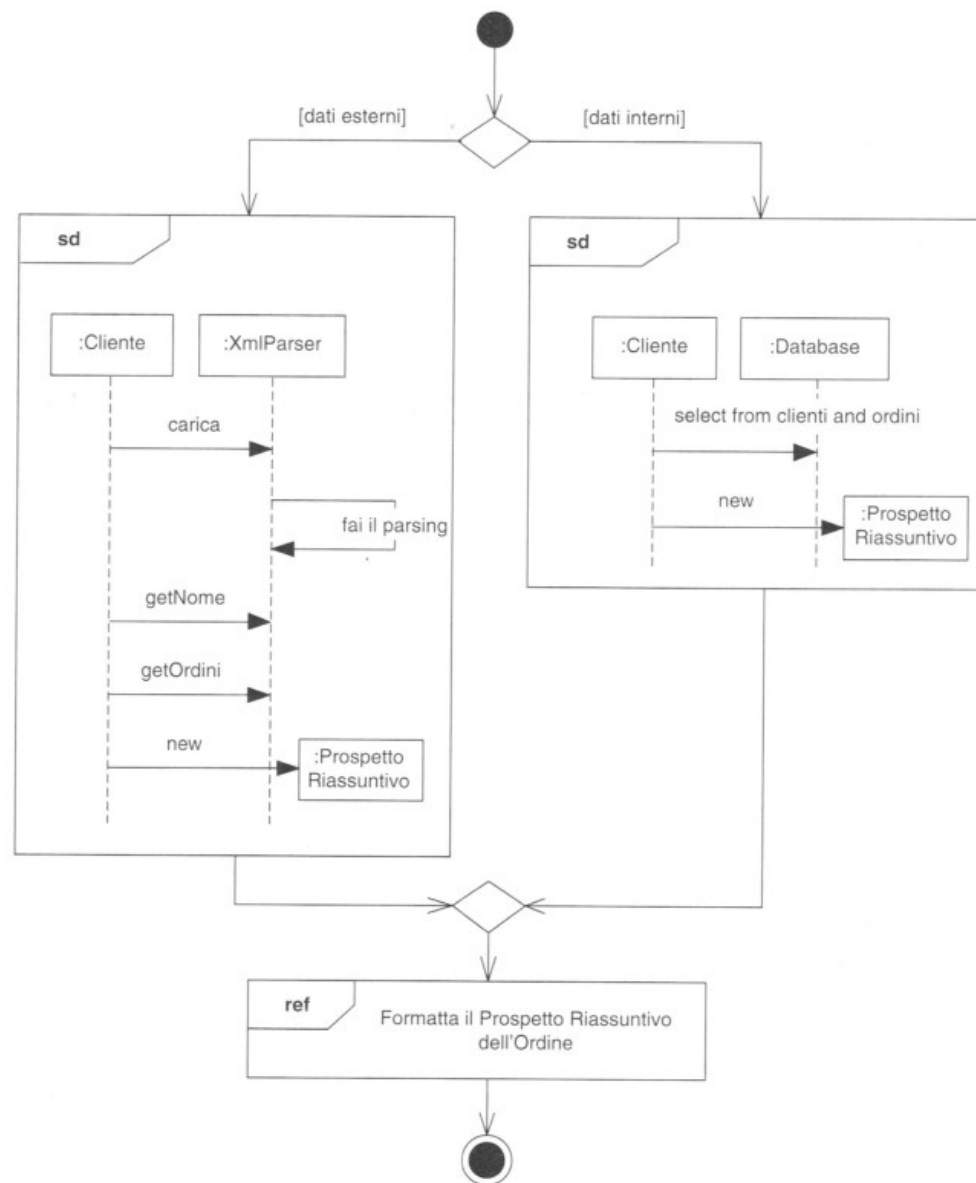


Diagramma di interazione generale (*interaction overview*)



Schede CRC (Classe-Responsabilità-Collaborazione)

- Non appartengono a UML
- Si usano per esplorare alternative di comportamento e di interazione delle classi con cui si intende implementare un caso d'uso (o i suoi scenari), prima di effettuare una scelta che sarà poi documentata coi diagrammi di interazione (usare questi ultimi per tale esplorazione sarebbe troppo pesante)

Scheda = cartoncino 10 x 15 cm

Nome_Classe		
Responsabilità		Collaborazione

Responsabilità di una classe

- È la descrizione di alto livello degli scopi (da uno a tre) per cui è stata creata la classe e dei doveri della stessa nei confronti delle altre e del sistema
- Permette di superare la visione della classe come semplice contenitore di dati
- Può corrispondere a un'operazione, a un attributo, o a un'aggregazione di attributi e operazioni

Collaboratori di una classe = altre classi con cui la classe interagisce

Ingegneria del Software

Settore dell'Informatica che si occupa di sistemi sw

- di dimensioni e complessità elevate
- realizzati da squadre
- disponibili in più versioni
- di lunga durata
- destinati a sottostare a cambiamenti

Implica una programmazione in grande ma ...

ingegneria del sw \neq programmazione

Consiste di molte altre attività in aggiunta alla programmazione

È una parte dell'ingegneria dei sistemi

Fase pionieristica (anni '40)

- Prime applicazioni = automazione di procedimenti di calcolo → calcolatore = strumento per l'esecuzione di operazioni numeriche che difficilmente potrebbero essere eseguite manualmente con la precisione richiesta
- Linguaggi / strumenti di basso livello
- Spesso sviluppatore del sw = utente del sw
- Applicazioni di vita breve

Dal calcolo alla elaborazione dati

- Avvento di applicazioni di natura gestionale →
calcolatore = macchina per creare, mantenere, modificare, distribuire informazione
- Distinzione tra sviluppatori e utenti
- Esigenza di una professionalità specifica → nascono EDP e sw house
- Dall'arte (lavoro individuale creativo) all'artigianato (lavoro di piccoli gruppi specializzati)
- Aumento della criticità e della complessità delle applicazioni
- Automazione di settori non tradizionali
- Tardi anni '50: avvento dei linguaggi di alto livello

Fase industriale (seconda metà anni '60)

- L'attività di sviluppo e manutenzione del sw coinvolge gruppi di lavoro, anche di grosse dimensioni, il cui lavoro deve essere pianificato e coordinato
- L'attività di progettazione del sw deve essere sempre meno sviluppata manualmente e informalmente e sempre più essere supportata da strumenti automatici
- Lo sviluppo deve seguire metodologie efficaci e deve aderire a *standard* di produzione che rendano la stessa insensibile al ricambio di personale
- Nuove applicazioni devono potere essere ottenute anche assemblando componenti standardizzati già sviluppati
- Produttività
- Qualità (deve essere certificata)

La crisi del sw

- I grandi progetti sw sono affetti da una incapacità cronica di rispettare i vincoli relativi a scadenze e budget
- Domanda di sw largamente insoddisfatta, ritardi fra il momento in cui sorge la richiesta di un'applicazione e il momento in cui la richiesta inizia a essere soddisfatta; cause:
 - ✓ Immaturità rispetto alle altre discipline ingegneristiche
Es. ingegneria civile: metodologie assestate che consentono una descrizione precisa e rigorosa dell'artefatto da progettare, modelli a diverso livello di astrazione, produzione di disegni dettagliati in base ai quali il sistema sarà realizzato, collaudo finale allo scopo di fornire una certificazione
 - ✓ Carenza numerica di personale specializzato
 - ✓ Grande quantità di addetti impiegati nella manutenzione

Gorgo del sw: carenza di personale specializzato → uso di personale impreparato/inesperto → produzione di sw di cattiva qualità → grande sforzo di manutenzione richiesto → impiego di una grande quantità di personale → carenza di personale

La crisi del sw (cont.)

- Rapporto

$$\frac{\text{costo sw}}{\text{costo hw}}$$

crescente; cause:

- ✓ la produzione di sw è scarsamente automatizzabile (attività brain-intensive)
- ✓ complessità delle applicazioni richieste

Si parla spesso di quella che viene definita ‘crisi del sw’, che però è solo apparentemente una crisi. Si tratta invece del fatto ben noto che numerosi prodotti sw in uso, che non sono stati progettati da ingegneri qualificati, siano di qualità più o meno scadente. Certo, questo sw è difficile da modificare ed è tecnicamente mal documentato. La dipendenza del funzionamento del sw dalla presenza di chi lo ha sviluppato è talmente grande che in molti casi lo si può considerare come il prodotto di un’attività amatoriale, certamente non come un lavoro professionale e tanto meno come esempio di tecnica e capacità ingegneristica (W. Zuser, S. Biffel, T. Grechenig, M. Köhle, 2001)

Quando un programma sw ha successo, ovvero quando risponde alle esigenze delle persone che lo usano, si comporta senza problemi in un lungo arco di tempo, è facile da modificare e ancora più facile da utilizzare, cambia in meglio la nostra vita.

Quando il sw fallisce gli obiettivi, ovvero quando gli utenti sono insoddisfatti, quando il sw è soggetto a errori, quando è difficile da modificare e ancora più difficile da utilizzare, si verificano varie situazioni negative.

Tutti noi vogliamo realizzare del sw che cambi in meglio il mondo, evitando tutto ciò che accade quando non si riesce a ottenere un buon risultato. Per ottenere ciò è necessario introdurre disciplina nella progettazione e nella realizzazione del sw. Questo è il motivo per cui è necessario un approccio ingegneristico

(Pressman, 2004)

Nasce una nuova disciplina

In una conferenza NATO tenuta a Garmisch nel 1968 viene coniato il termine “Ingegneria del Software” a testimoniare l’esigenza di una disciplina ingegneristica, basata su solide basi teoriche e metodologiche che permettano la progettazione, produzione e manutenzione di applicazioni che forniscano caratteristiche di qualità prefissate mediante l’uso delle risorse previste

L’oggetto non è la computazione e la teoria sottostante ma piuttosto l’uso dell’informatica per risolvere dei problemi (chimico vs. ing. chimico)

Programmi sw = prodotti industriali di supporto ad altri settori produttivi

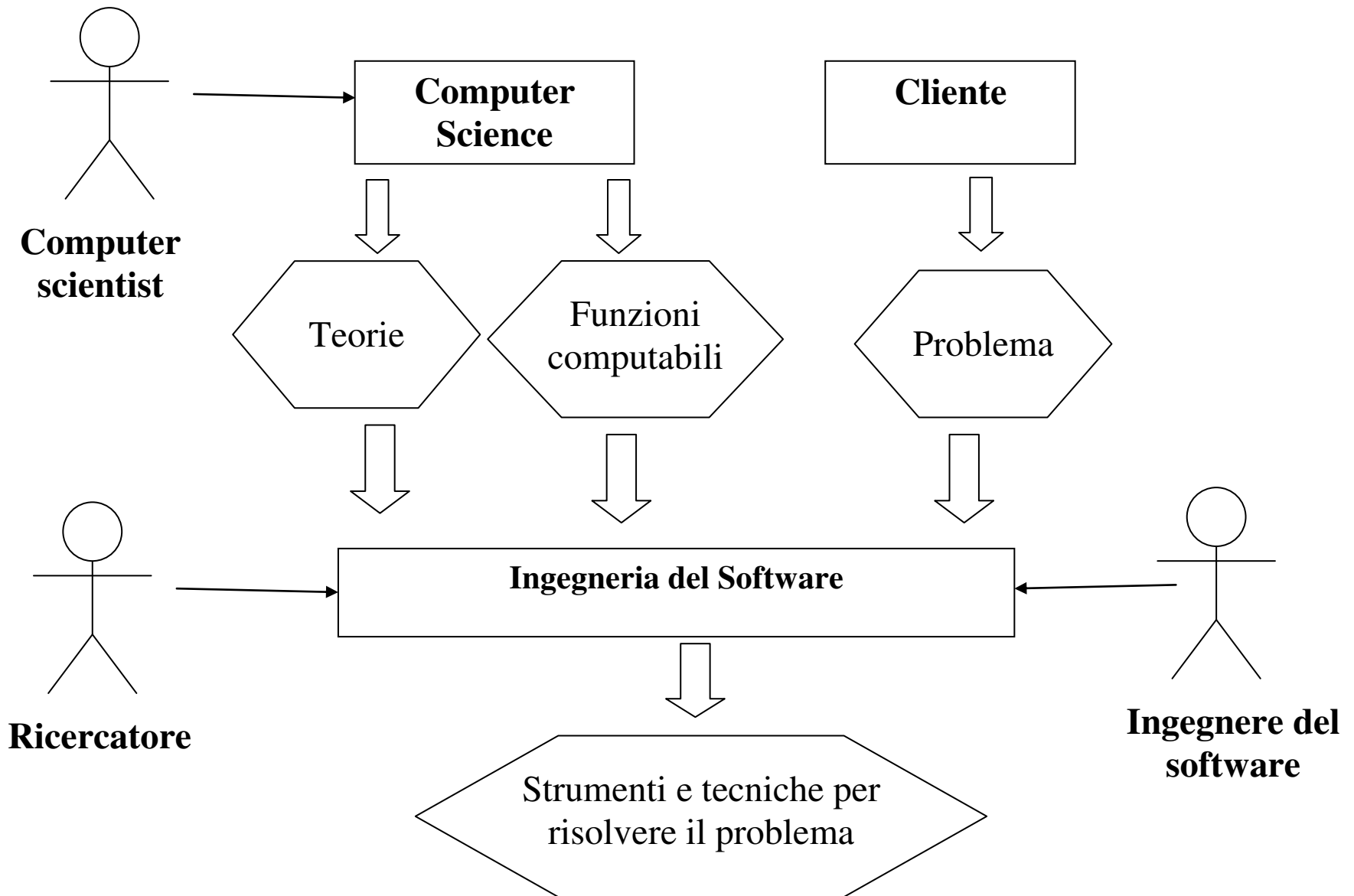
IEEE Standard 610.12-1990 - Definizione di Ingegneria del Software

- (1) Applicazione di un approccio sistematico, disciplinato e misurabile allo sviluppo, esercizio e manutenzione del software; cioè applicazione dell'ingegneria al sw;
- (2) Studio di strategie di supporto al punto precedente.

Parnas – 1978 - Definizione di Ingegneria del Software

Costruzione da parte di più persone di software disponibile in più versioni

Relazione tra Software Engineering e Computer Science



Relazioni tra

Software Engineering e Computer Science

Linguaggi di programmazione

Sistemi operativi

Basi di dati

Intelligenza artificiale

Informatica teorica

Software Engineering e altre discipline

Ingegneria gestionale

Ingegneria dei sistemi

Caratteristiche esclusive del sw

- Malleabilità apparente (sembra facile modificare il sw ma non è facile operare modifiche al fine di ottenere un certo comportamento desiderato)
- Immaterialità \Rightarrow solo progettazione
- Intangibilità (difficoltà di descrizione e valutazione)
- Assenza del processo di assemblaggio
- Evoluzione tecnologica rapidissima

Processo produttivo del sw (= ciclo di vita): dove termina lo *sviluppo* di una applicazione e dove inizia la sua *evoluzione*?

Approccio ingegneristico alla produzione del sw

- Mira a garantire alti livelli di controllo sulla qualità grazie a un processo formale che descrive le varie fasi che vanno seguite nello sviluppo del sw
- Molte persone lavorano allo stesso progetto, quindi la documentazione è importante (vedi progetti dell'ingegneria civile): “disegno” di alto livello e di dettaglio
- La fase iniziale di raccolta dei requisiti del cliente prevede che siano esibiti diagrammi, prototipi e documenti tali da garantire che ciò che si svilupperà è effettivamente ciò che vuole il cliente
- Centralità della progettazione, fondamentale la componente intellettuale creativa
- Il testing del prodotto segue un processo ben formalizzato, in cui i requisiti iniziali del cliente sono riesaminati per verificarne la corretta implementazione

L'ingegnere del sw

Alle diverse fasi corrispondono figure professionali diverse (suddivisione di competenze e ruoli); nei piccoli progetti 2 o 3 persone coprono tutti i ruoli:

- analista
- progettista
- programmatore
- ingegnere del testing
- istruttore (mostra agli utenti come usare il sistema)
- gruppo manutenzione (include analisti, progettisti, ...)
- gruppo controllo qualità
- librarian (redattore di documenti usati durante la vita del sistema, ad es. specifiche dei requisiti, descrizioni di progetto, documentazione di programma, manuali di addestramento, ecc.)
- squadra di gestione delle configurazioni (documenta le corrispondenze fra requisiti, progetto, implementazione e test, ad es. per poter dire ai manutentori quale funzione modificare se è richiesto un cambiamento nei requisiti, ecc.; inoltre coordina le diverse versioni di un sistema)

Diagrammi di comunicazione

- Diagrammi di interazione che hanno la stessa portata informativa dei diagrammi di sequenza (frame di interazione esclusi)
- Rispetto ai diagrammi di sequenza sono più facili da disegnare su una lavagna ma la loro lettura è meno immediata
- Enfatizzano il collegamento fra oggetti
- La disposizione delle icone che rappresentano gli oggetti può assumere un particolare significato per il progettista (ad es. la suddivisione in package)

Diagramma di comunicazione

Elementi	Sintassi	Semantica
Oggetto (partecipante)	Scatola contenente <i>NomeOggetto</i> <i>:NomeClasse</i> , dove sia <i>NomeOggetto</i> , sia <i>:NomeClasse</i> , sono opzionali ma non possono mancare entrambi	Indica un'istanza di una classe
Collegamento fra due oggetti	Linea continua tesa fra due scatole (non necessariamente distinte) Se la scatola sorgente e quella destinazione coincidono si parla di autocollegamento Il collegamento è sospeso se proviene dalla fonte esterna (non visualizzata) che invia il msg trovato	Gli oggetti collegati collaborano nell'ambito dell'interazione descritta dal diagramma

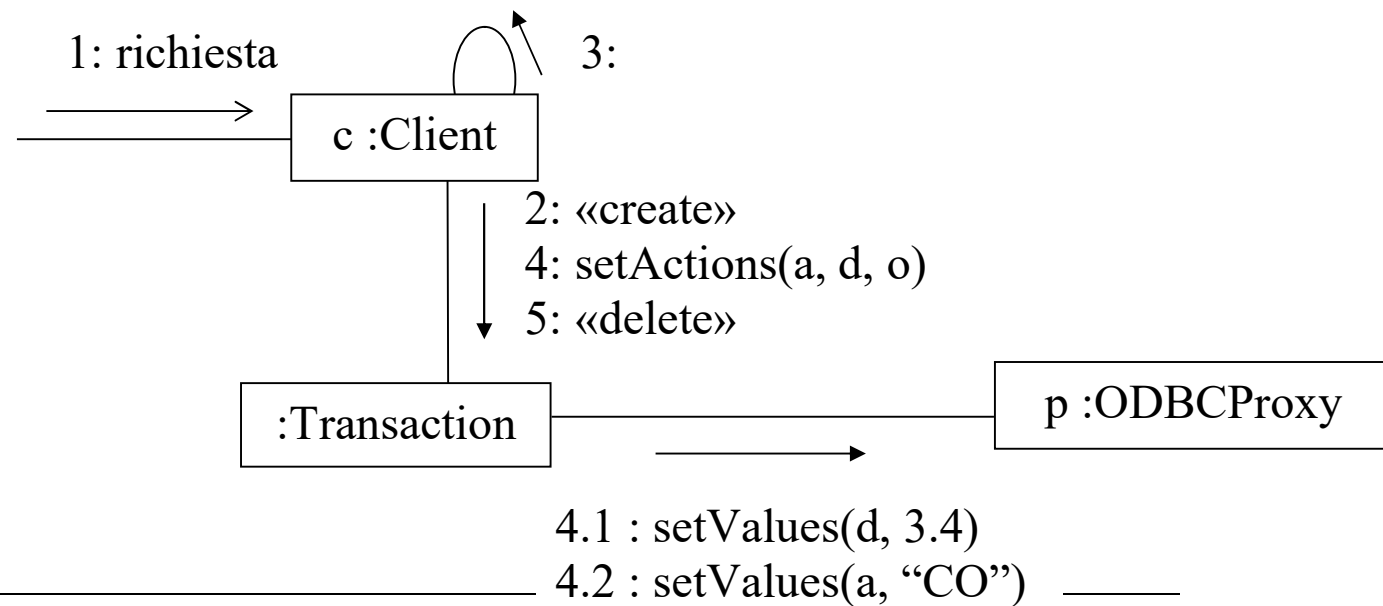
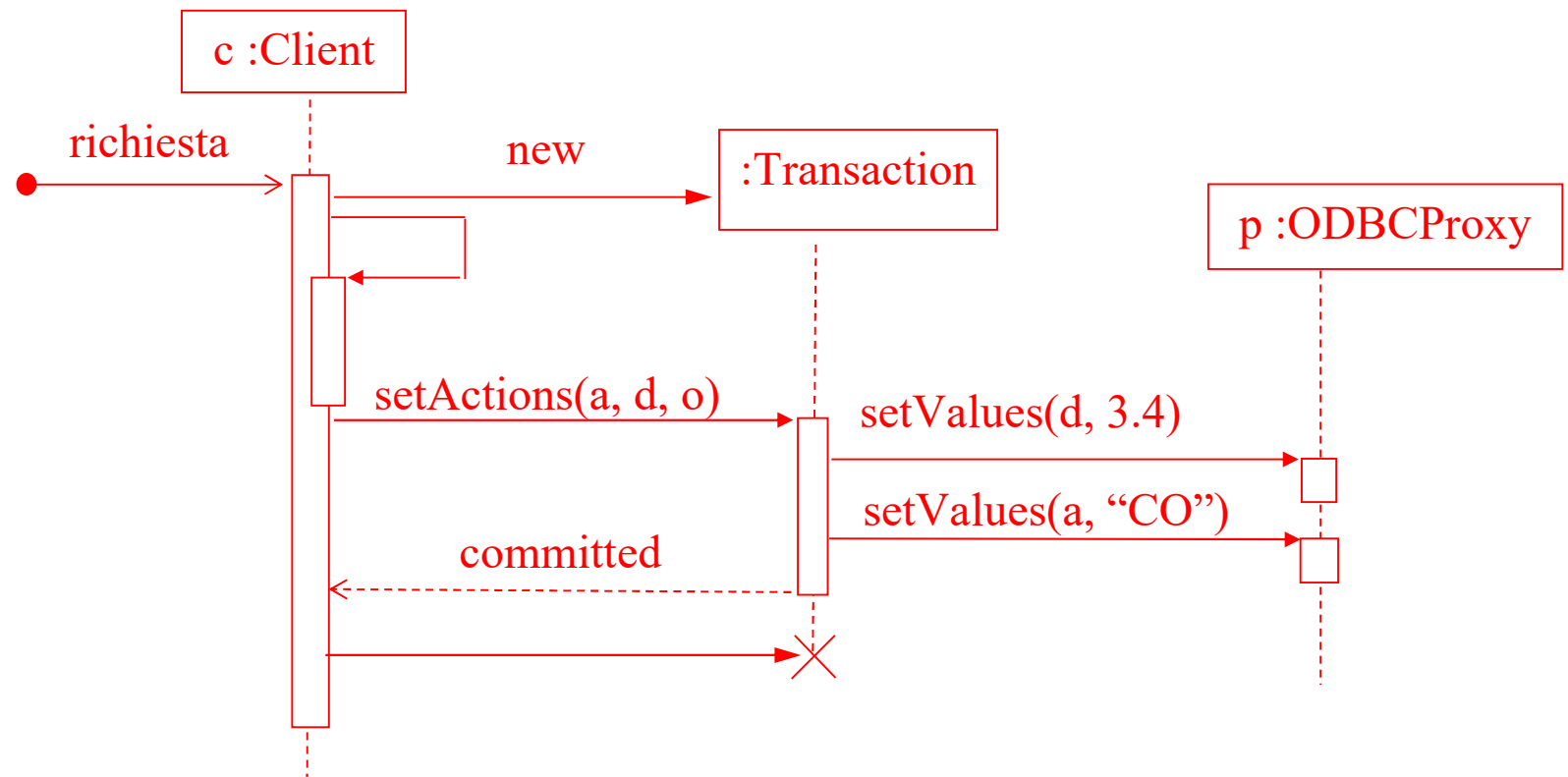
Diagramma di comunicazione (cont.)

Elementi	Sintassi	Semantica
Messaggio	<ul style="list-style-type: none">• Freccia affiancata a un collegamento• Una singola freccia può rappresentare più msg, ciascuno numerato e fornito di nome analogo a quello dei diagrammi di sequenza, tutti aventi lo stesso verso• La numerazione è decimale nidificata (es. 1.1, 1.2, ecc.), per rappresentare invocazioni innestate• La numerazione può contenere anche una lettera (es. A5) in modo da distinguere messaggi appartenenti a thread diversi• Ciascun msg può essere preceduto da indicatore di iterazione e/o guardia• Affiancate a un collegamento (non sospeso) possono esserci due frecce di verso opposto	La numerazione indica la sequenza totale dei messaggi all'interno del diagramma

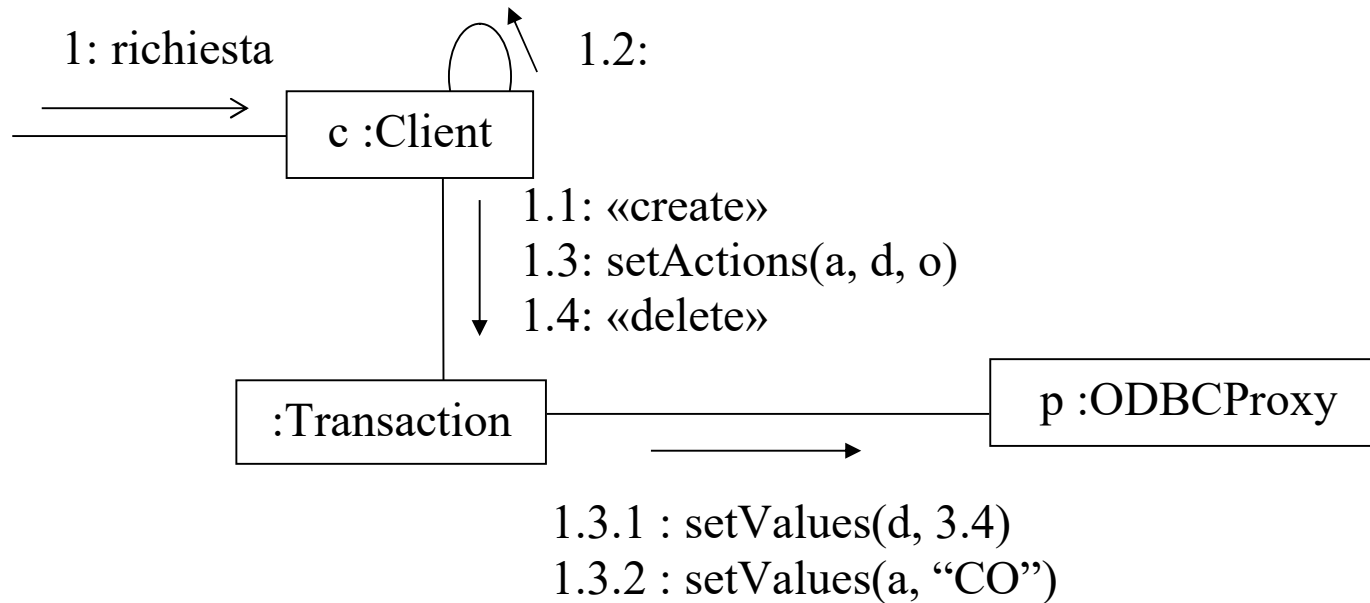
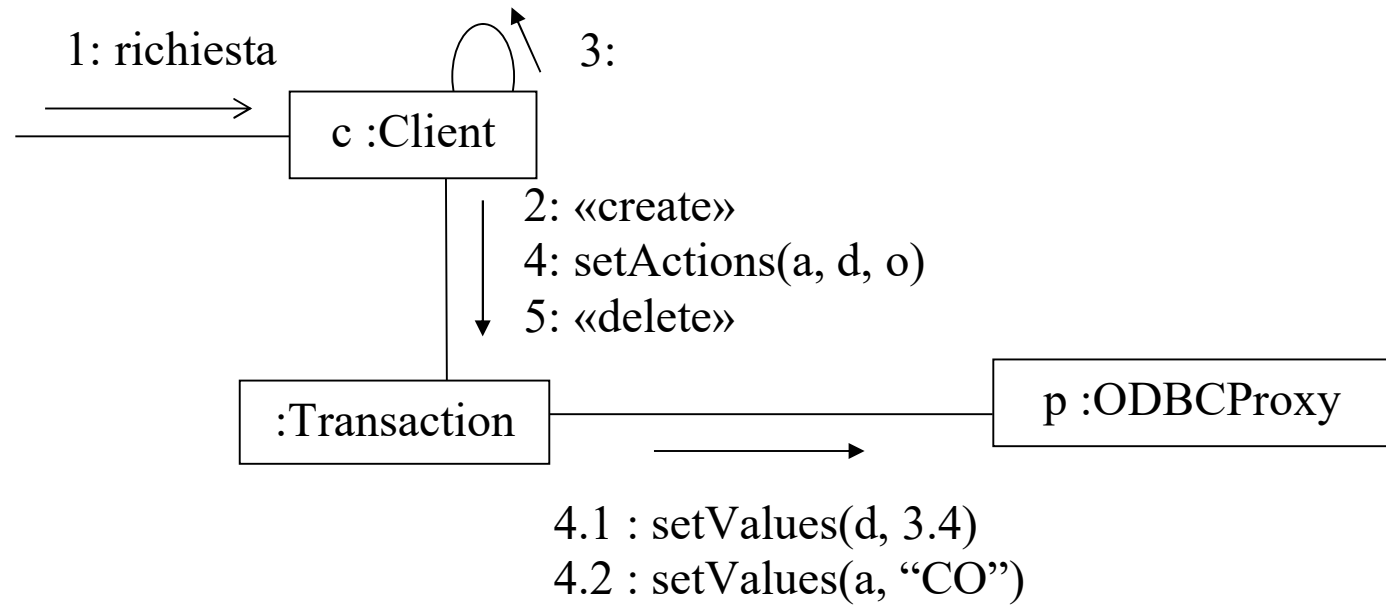
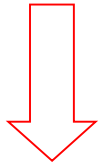
Diagramma di comunicazione (cont.)

Elementi	Sintassi	Semantica
Indicatore di iterazione	Asterisco (eventualmente accompagnato dalla base dell'iterazione fra parentesi quadre)	Il msg a cui l'indicatore si riferisce viene inviato più volte Notazione insufficiente a specificare completamente la logica di controllo (ad es. il msg viene inviato più volte a un unico oggetto o a diversi oggetti, tutti dello stesso tipo ?)

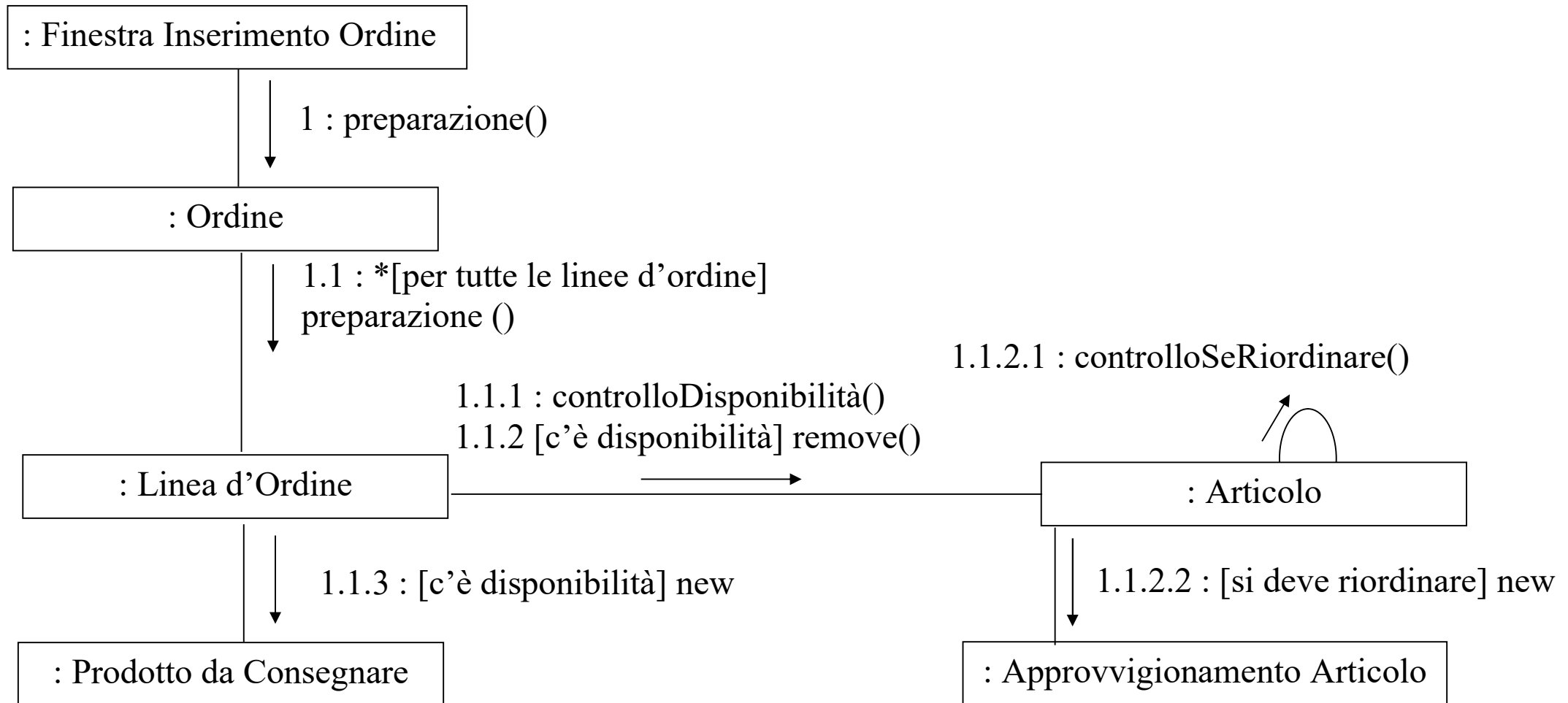
Esempi (1)



Numerazione normativa



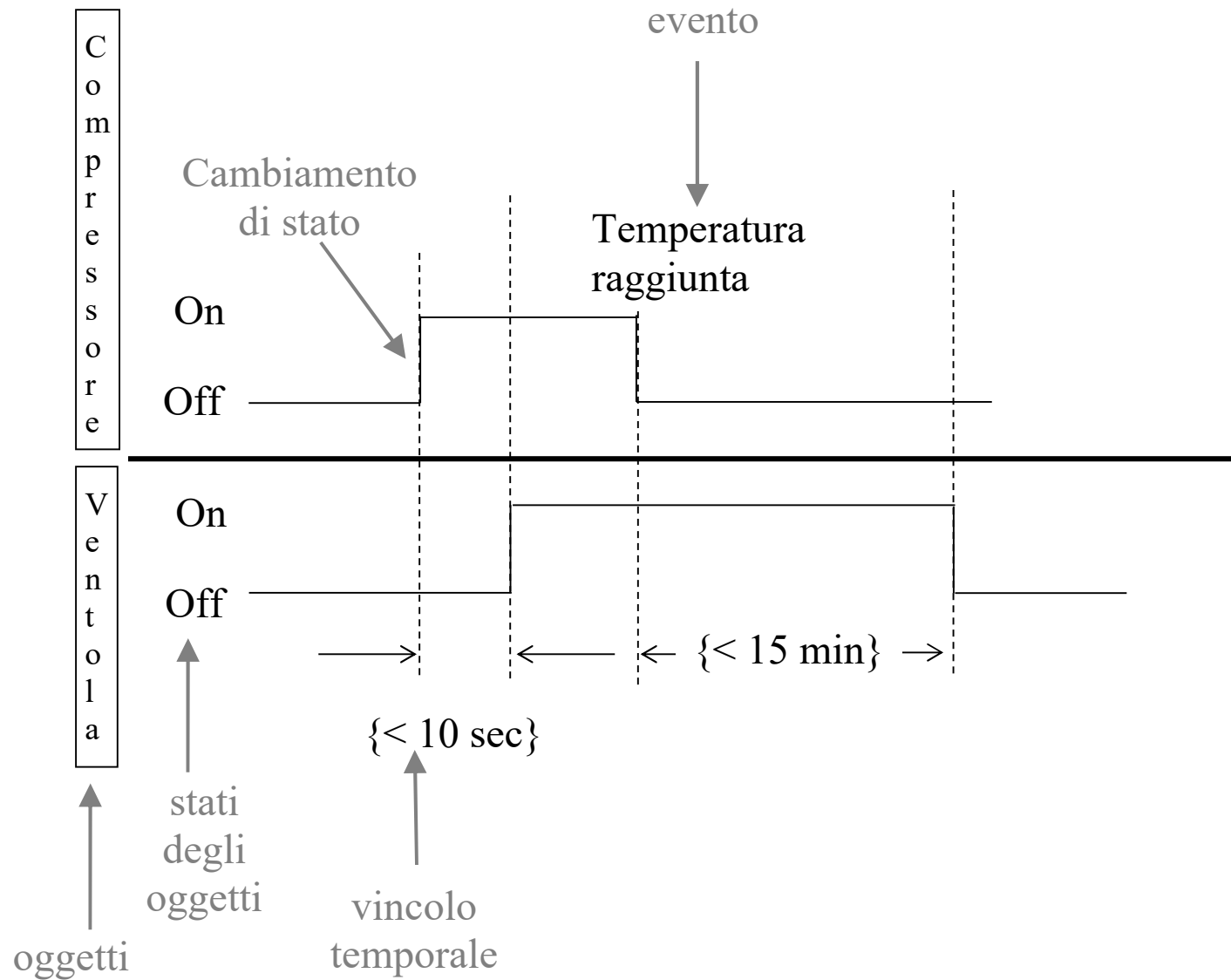
Esempi (2)



Diagrammi di temporizzazione

- Forma dei diagrammi di interazione dedicata all'espressione dei vincoli temporali tra i cambiamenti di stato di oggetti differenti
- Esistevano già in elettronica

Esempio




Diagrammi di deployment

- Sono molto usati
- Documentano la distribuzione fisica di un sistema, mostrando i vari pezzi sw in esecuzione su una o più macchine
- Sono utili soprattutto se la “messa in opera” del sistema non è estremamente semplice

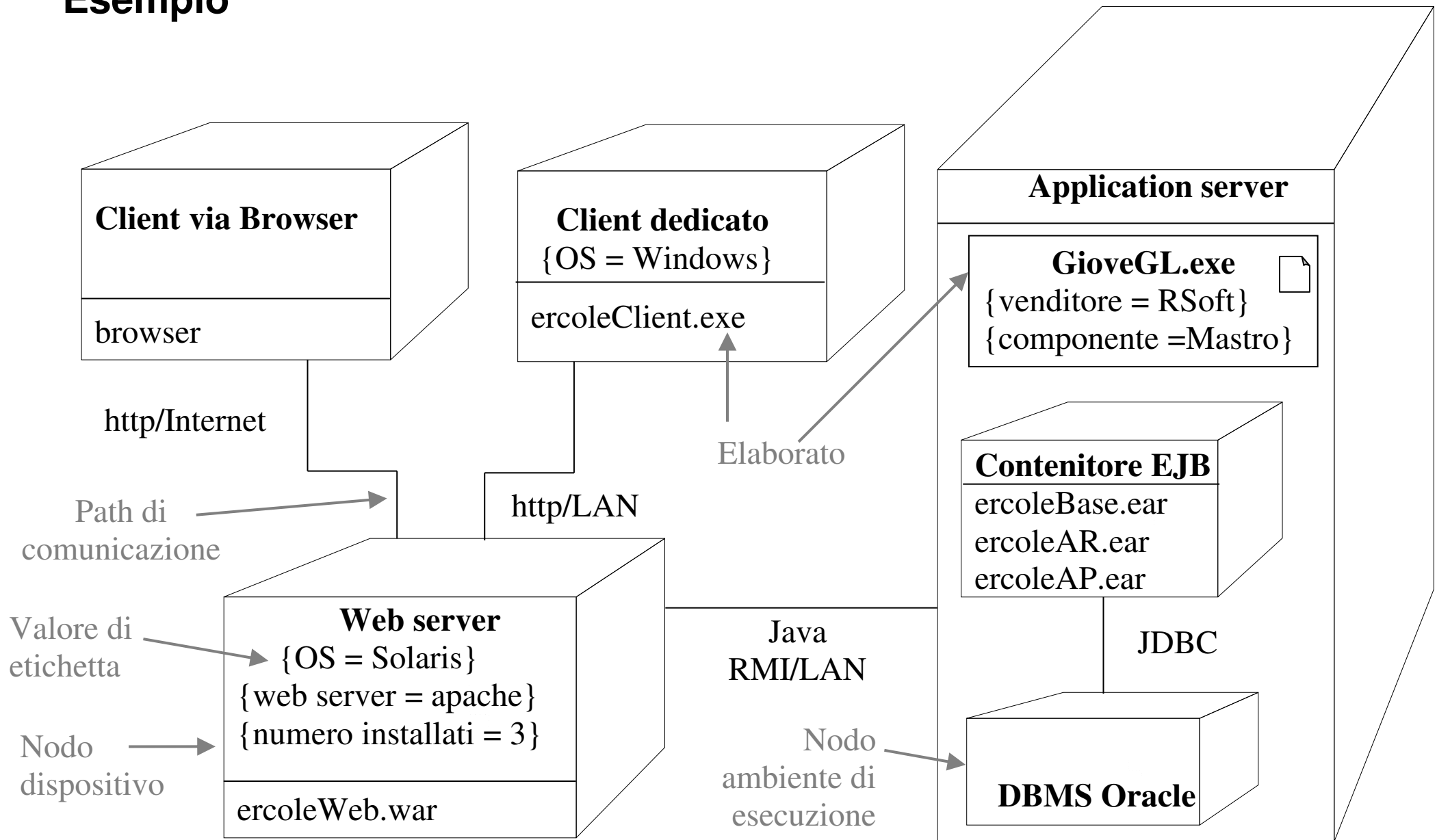
Diagramma di deployment

Elementi	Sintassi	Semantica
Nodo	<p>Parallelepipedo suddiviso orizzontalmente in due sezioni:</p> <ul style="list-style-type: none">• la sezione superiore contiene il nome del nodo in grassetto ed eventuali vincoli (tagged value),• la sezione inferiore contiene eventuali elaborati e/o un diagramma di deployment rappresentante del sw in esecuzione all'interno del nodo stesso	<p>Unità che consente l'esecuzione del sw; può essere</p> <ul style="list-style-type: none">• un <u>dispositivo</u> (<i>device</i>), cioè un pezzo hw (che può essere un computer completo o un componente hw più semplice)• un <u>ambiente di esecuzione</u>, cioè un pezzo sw che può contenere ed eseguire altro sw (es. sistema operativo)
Path di comunicazione	<p>Linea continua che connette due nodi, opzionalmente dotata di etichetta (stringa)</p>	<p>Flusso di comunicazione</p> <p>L'etichetta può aggiungere info, ad es. circa i protocolli utilizzati</p>

Diagramma di deployment (cont.)

Elementi	Sintassi	Semantica
Elaborato	<p>Nome (stringa) oppure box di classe, dotato di icona  o della parola chiave</p> <p>«artifact»</p>	<p>File (.exe binario, DLL, JAR, assembler, script, di dati, di configurazione, HTML, ecc.)</p> <p>Il nodo entro cui si trova indica la sua posizione nel sistema durante l'esecuzione</p> <p>Spesso rappresenta l'implementazione di un componente (come documentabile attraverso un valore di etichetta)</p>
Valore di etichetta (<i>tagged value</i>)	<p>Stringa racchiusa fra parentesi graffe associata a un nodo o a un elaborato</p> <p>{ tag = valore }</p>	<p>Info aggiuntiva interessante da documentare (es. nome del sistema operativo, modello del computer)</p> <p>All'interno di un nodo può anche indicare il numero di esemplari di quel nodo presenti nel sistema (senza disegnare più nodi distinti)</p>

Esempio

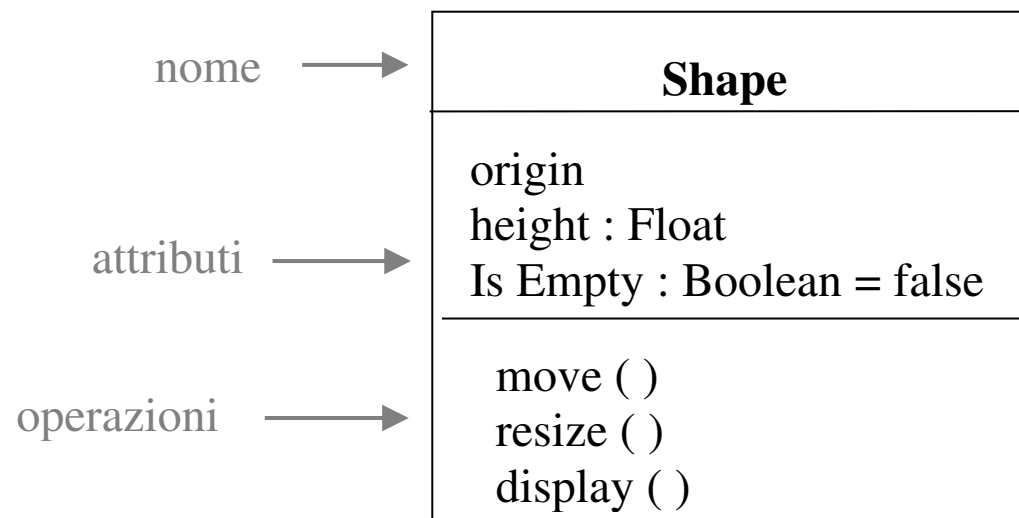


Diagrammi delle classi

- Tecnica centrale di modellazione OO
- Descrizione strutturale statica degli oggetti che compongono il sistema (comprensiva di attributi e operazioni) e delle loro relazioni (restrizioni incluse)
- Descrizione dei vincoli
- Sono simili ai modelli per i dati → è facile basare erroneamente il loro sviluppo sui dati piuttosto che sulle responsabilità

Concetti essenziali

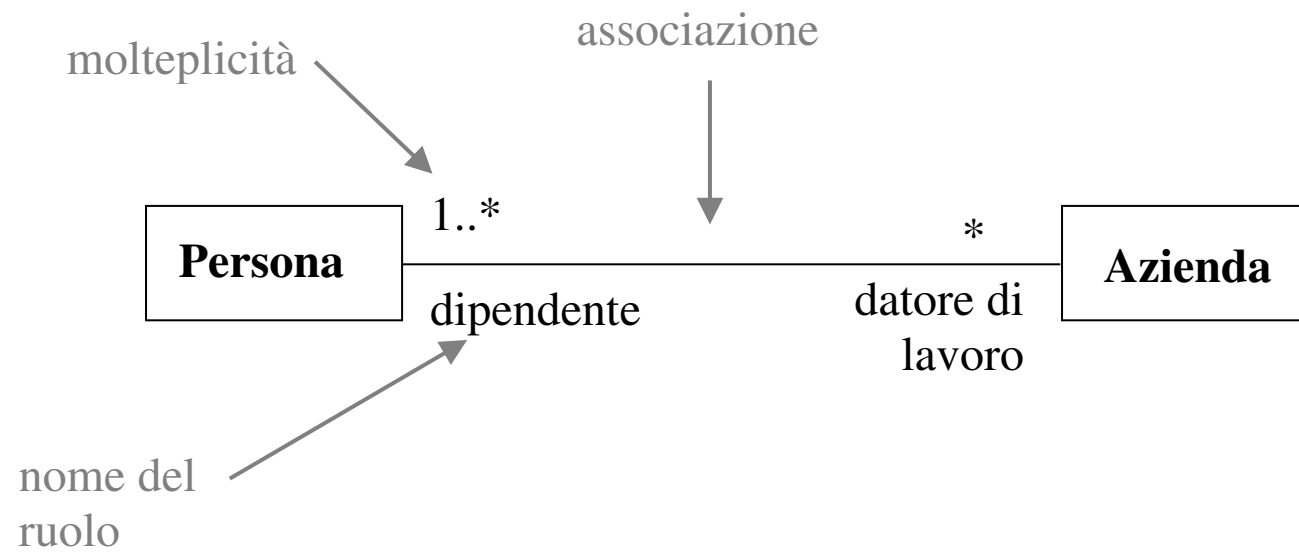
Elementi	Sintassi	Semantica
Classe	<p>Scatola suddivisa in tre parti orizzontali, contenenti rispettivamente <i>nome</i>, <i>attributi</i> (lo stato) e <i>operazioni</i> (il comportamento);</p> <p><i>nome</i> è una stringa in grassetto che identifica univocamente la classe e può essere preceduta dal nome del package che contiene la classe (es. java::awt::Rectangle); la sezione del <i>nome</i> è l'unica obbligatoria</p>	Insieme di oggetti che condividono gli stessi attributi, operazioni, relazioni e semantica



Concetti essenziali (cont.)

Elementi	Sintassi	Semantica
Attributo	<p><i>visibilità nome: tipo</i> <i>[molteplicità] = valore-di-default {stringa-elenco-proprietà-aggiuntive}</i> dove solo <i>nome</i>, che è una stringa, è obbligatorio</p> <p>Es. - titolo: String [1] = "UML distilled" {readOnly}</p>	<p><i>default</i> = valore in un oggetto appena creato, se non specificato diversamente durante la creazione</p> <p>Se la stringa delle proprietà aggiuntive non è presente, generalmente l'attributo è modificabile</p>
Visibilità (di attributi e operazioni)	<p>+ (pubblica) - (privata) # (protected) ~ (package)</p>	<ul style="list-style-type: none"> • Gli elementi pubblici possono essere usati da un'altra classe, quelli privati invece sono riservati all'uso interno • Per la semantica precisa degli identificatori bisogna fare riferimento alle regole del linguaggio di programmazione adottato

Associazione

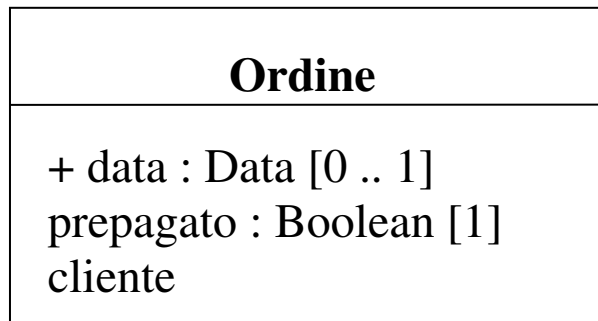


Concetti essenziali (cont.)

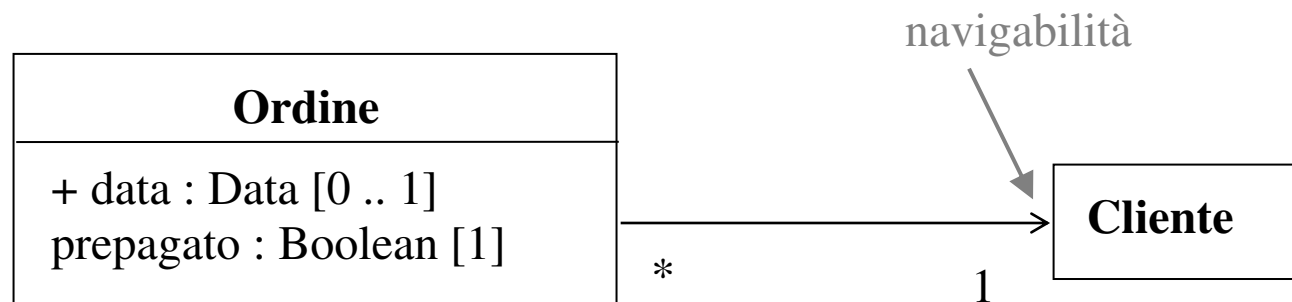
Elementi	Sintassi	Semantica
Associazione (o ruolo)	<p>Linea continua che unisce due classi; opzionalmente ciascun capo della linea può terminare con una punta biforcuta, che indica la <i>navigabilità</i>, e può essere etichettato con:</p> <ul style="list-style-type: none">• <i>molteplicità</i> (es. 1, *, 0..1)• <i>nome del ruolo</i> (stringa da specificare solo quando fa aumentare la comprensibilità, altrimenti il nome implicito del ruolo è quello della classe attaccata al capo)	<ul style="list-style-type: none">• Relazione statica fra le istanze di due classi;• la <i>molteplicità</i> è il numero di oggetti che prendono parte a tale relazione;• la <i>navigabilità</i> è il verso di percorribilità del collegamento (e non è rilevante nel modello concettuale);• l'assenza dell'indicazione di navigabilità si interpreta o come mancanza di info circa la stessa o come associazione bidirezionale, che è difficile da implementare perché richiede che le due proprietà siano sempre “sincronizzate”

Proprietà di una classe

Proprietà espressa come attributo



Proprietà espressa come associazione



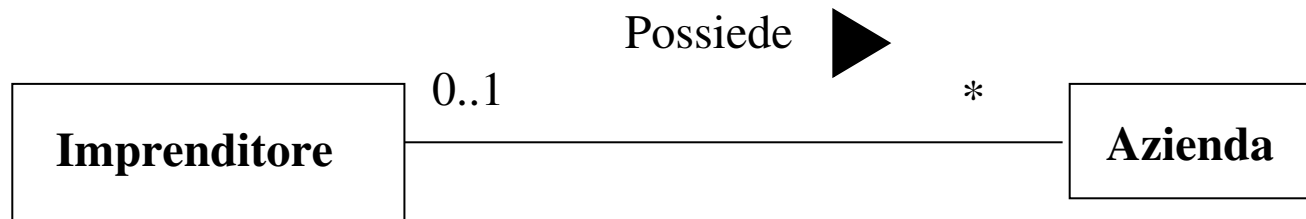
Molteplicità delle proprietà

- Si indicano gli estremi inferiore e superiore di un intervallo, come 2..4, dove * rappresenta un valore illimitato
- 1 equivale a 1..1
- * è un'abbreviazione per 0..*
- Se una proprietà ha più valori, è preferibile indicare il suo nome in forma plurale
- Gli elementi di una molteplicità a più valori formano un insieme; se ciò non è accettabile, è necessario dare un'indicazione diversa come {ordered}, {nonunique} o {bag}

Molteplicità delle proprietà

Elementi	Sintassi	Semantica
Vincoli circa una molteplicità a più valori	<code>{ordered}</code> <code>{nonunique}</code> <code>{bag}</code> <code>{hierarchy}</code> <code>{dag}</code>	<p>Possono coesistere anche più vincoli (purché matematicamente consistenti)</p> <ul style="list-style-type: none">• <code>{ordered}</code>: esiste un ordinamento fra gli elementi• <code>{nonunique}</code>: gli elementi possono comparire più volte• <code>{bag}</code>: gli elementi formano un multinsieme• <code>{hierarchy}</code>: gli elementi formano una gerarchia• <code>{dag}</code>: gli elementi formano un DAG (grafo orientato aciclico)

Verbo come nome di associazione



Diagrammi delle classi: concetti essenziali (cont.)

Elementi	Sintassi	Semantica
Operazione	<p><i>visibilità nome (lista-parametri): tipo-ritornato {stringa-proprietà}</i> dove</p> <ul style="list-style-type: none"> • <i>nome</i> è una stringa, • <i>lista-parametri</i> è una serie di parametri, ciascuno specificato mediante un elemento <i>direzione nome: tipo = valore-di-default</i> e separato dal successivo mediante una virgola (la direzione può assumere i valori <i>in</i>, <i>out</i>, <i>inout</i>, se manca si presume sia <i>in</i>), • <i>stringa-proprietà</i> indica i valori delle proprietà dell'operazione, cioè <i>query</i>, <i>modificatore</i>, <i>get</i> e <i>set</i> 	<p>Azione che una classe sa come eseguire;</p> <ul style="list-style-type: none"> • se un'operazione <i>i</i>) non modifica lo stato del sistema e <i>ii</i>) ritorna dei valori è una <i>query</i>, • se un'operazione <i>i</i>) modifica lo stato osservabile del sistema e <i>ii</i>) non ritorna valori è un <i>modificatore</i>; • un metodo <i>get</i> è una particolare <i>query</i> che restituisce il valore di un attributo e non fa nient'altro; • un metodo <i>set</i> è un particolare <i>modificatore</i> che assegna il valore a esso passato a un attributo e non fa nient'altro <p>Una convenzione comune è cercare di scrivere modificatori che non restituiscano mai valori</p>

Operazioni

- Quelle che manipolano le proprietà della classe di solito si possono dedurre, per cui non sono incluse nel diagramma
- Nei modelli concettuali, dovrebbero indicare le principali responsabilità della classe, magari facendo riferimento a una carta CRC
- Corrispondono alla dichiarazione di una procedura e, quindi, si distinguono (sottilmente) dai metodi, che invece corrispondono al corpo della procedura → le due cose sono diverse in presenza di polimorfismo

Responsabilità di una classe = obbligo che la classe si assume nello svolgere un servizio. Fa parte del contratto che essa stipula con le altre classi

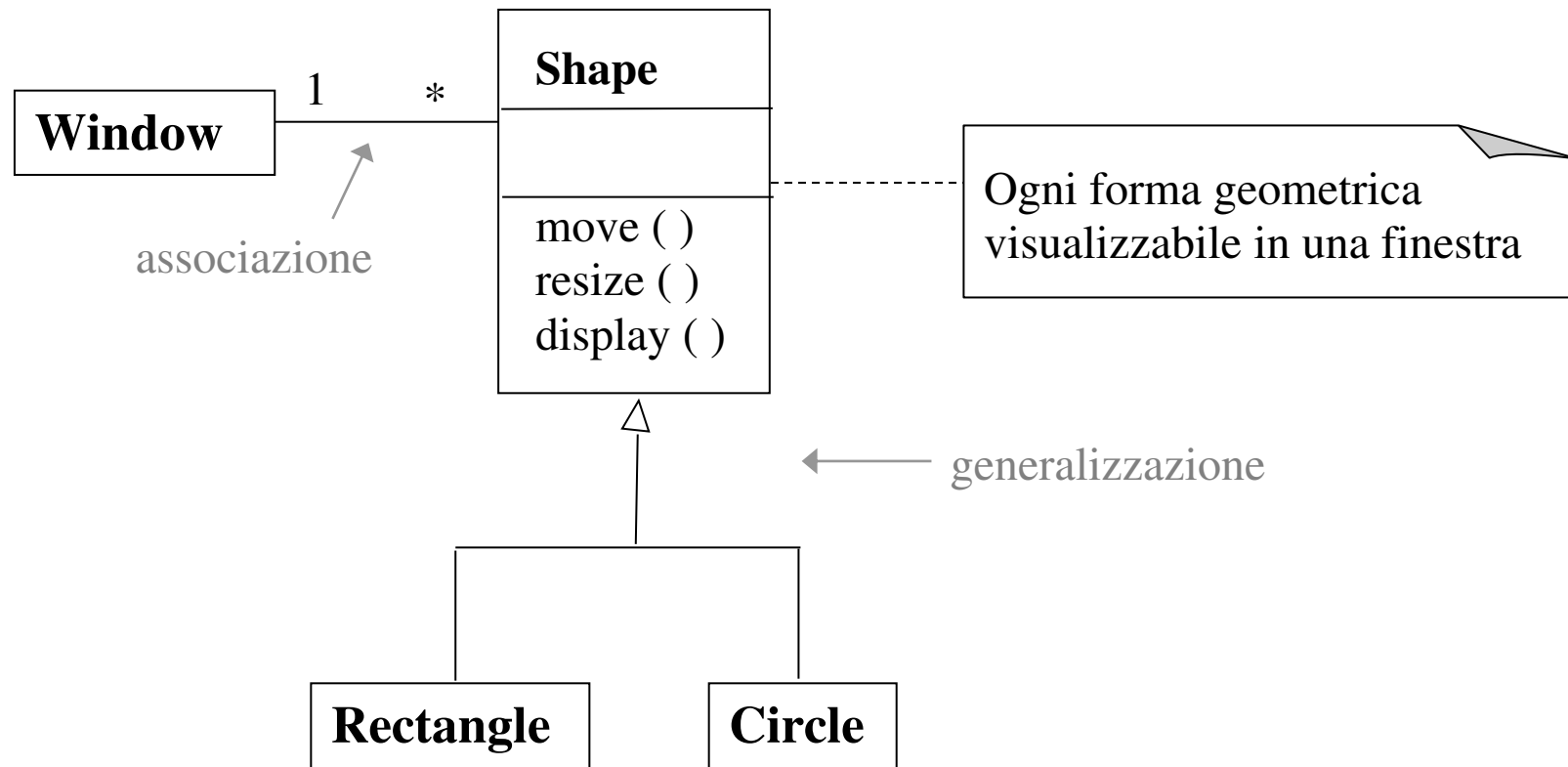
Concetti essenziali (cont.)

Toolbar
currentSelection: Tool # toolCount: Integer
+ pickItem(i: Integer) + addTool(t: Tool) + removeTool(i: Integer) + getTool(): Tool # checkOrphans() - compact()

Concetti essenziali (cont.)

Elementi	Sintassi	Semantica
Generalizzazione	Freccia con testa vuota e uno o più capi: la testa termina sulla superclasse, i capi si dipartono ciascuno da una sottoclasse	Se due o più classi sono diverse ma mostrano anche molte somiglianze, queste si possono raccogliere in una superclasse (relazione IS-A)
Nota	<p>Testo</p> <ul style="list-style-type: none">• che segue due trattini -- all'interno di un elemento del diagramma, oppure• contenuto in una scatola con “orecchietta”, eventualmente collegata all'elemento a cui si riferisce mediante una linea tratteggiata	<p>Commento aggiuntivo</p> <p>Una o più note possono apparire in qualsiasi tipo di diagramma UML</p> <p>Una convenzione comune prevede di mettere un cerchietto vuoto all'estremità della linea tratteggiata, per meglio collegarla all'elemento a cui si riferisce</p>

Concetti essenziali (cont.)



Prospettive e diagrammi delle classi

Prospettiva	Interpretazione
<i>Concettuale</i> (da adottarsi per realizzare il modello di dominio nella fase di elaborazione del RUP)	classe = concetto nella mente del cliente (→ il diagramma diventa il linguaggio dell'interlocutore) associazione = relazione fra concetti (la navigabilità non è utile) attributo = proprietà degli oggetti della classe, notazione alternativa rispetto all'associazione operazione = una delle responsabilità principali della classe generalizzazione: qualsiasi cosa si possa dire del supertipo (in termini di associazioni, attributi e operazioni), si può dire anche dei sottotipi

N.B. Le responsabilità di una classe possono essere descritte ciascuna mediante una nota inserita entro il riquadro della classe

Prospettive e diagrammi delle classi (cont.)

Prospettiva	Interpretazione
<i>Specifica</i>	<p>classe = tipo (un tipo è l'interfaccia – o parte di essa – di una o più classi “fisiche”, una classe “fisica” può implementare più tipi)</p> <p>associazione = responsabilità (indipendenti dalla struttura dati) ovvero, per ogni relazione R fra due classi C1 e C2 con navigabilità $C1 \rightarrow C2$, esistenza di un'interfaccia (implicita, che, a livello implementativo può essere realizzata in vario modo) che</p> <ul style="list-style-type: none">• dato un elemento di C1, determina tutti gli elementi di C2 che partecipano a R• effettua l'aggiornamento della relazione

Prospettive e diagrammi delle classi (cont.)

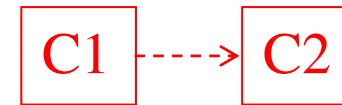
Prospettiva	Interpretazione
<i>Specifica (cont.)</i>	<p>attributo = responsabilità: esiste un'interfaccia (implicita) che</p> <ul style="list-style-type: none">• consente di impostare il valore dell'attributo di un oggetto• restituisce il valore dell'attributo <p>operazione = metodo pubblico</p> <p>generalizzazione = creazione di sottotipi (o ereditarietà di interfacce): l'interfaccia del sottotipo deve includere tutti gli elementi dell'interfaccia del supertipo (si dice che deve essere conforme all'interfaccia del supertipo);</p> <p>deve essere possibile sostituire un'istanza della sottoclasse all'interno di qualsiasi pezzo di codice che preveda l'esistenza di un'istanza della superclasse e tutto deve continuare a funzionare</p> <p>N.B. la creazione di sottoclassi “fisiche” è solo uno dei modi per implementare la creazione di sottotipi</p>

Prospettive e diagrammi delle classi (cont.)

Prospettiva	Interpretazione
<i>Implementazione</i>	<p>classe = classe “fisica”</p> <p>associazione $R (C1 \rightarrow C2)$ = dato un qualsiasi oggetto di $C1$ esistono specifici meccanismi di collegamento verso gli oggetti corrispondenti di $C2$</p> <p>la navigabilità può essere diversa da quella di specifica</p> <p>attributo = campo</p> <p>generalizzazione: la sottoclasse eredita tutti i metodi e i campi della superclasse e può ridefinirne le funzioni (eredità di implementazione)</p>

Dipendenza

Un elemento (detto *client*) di un diagramma (di qualsiasi tipo) dipende da un altro (detto *supplier*, cioè fornitore) se la modifica della definizione del secondo può causare un cambiamento del primo



Una classe C1 dipende da una classe C2 se

- invoca operazioni di C2 (anche solo un costruttore), e/o
- un suo campo è di tipo C2, e/o
- un parametro di una sua operazione è di tipo C2, e/o
- accede ai campi di C2

Man mano che il sistema cresce, il problema del controllo delle dipendenze aumenta

Specificare le dipendenze rende espliciti gli effetti a catena del cambiamento di un elemento → il sistema è più facilmente modificabile

Concetti essenziali (cont.)

Elementi	Sintassi	Semantica
Dipendenza	Freccia con linea tratteggiata e punta biforcuta	<p>La modifica della classe destinazione (fornitore) può causare un cambiamento della classe sorgente (client)</p> <p>Può legare pure una classe e un'interface (o una classe astratta), dove se l'interface (o la classe astratta) dovesse cambiare, anche la classe potrebbe cambiare</p>
Vincolo	Descrizione in linguaggio naturale racchiusa fra parentesi graffe, oppure descrizione in OCL	OCL (Object Constraint Language) fa parte di UML ed è basato sul calcolo dei predicati

Dipendenza (cont.)



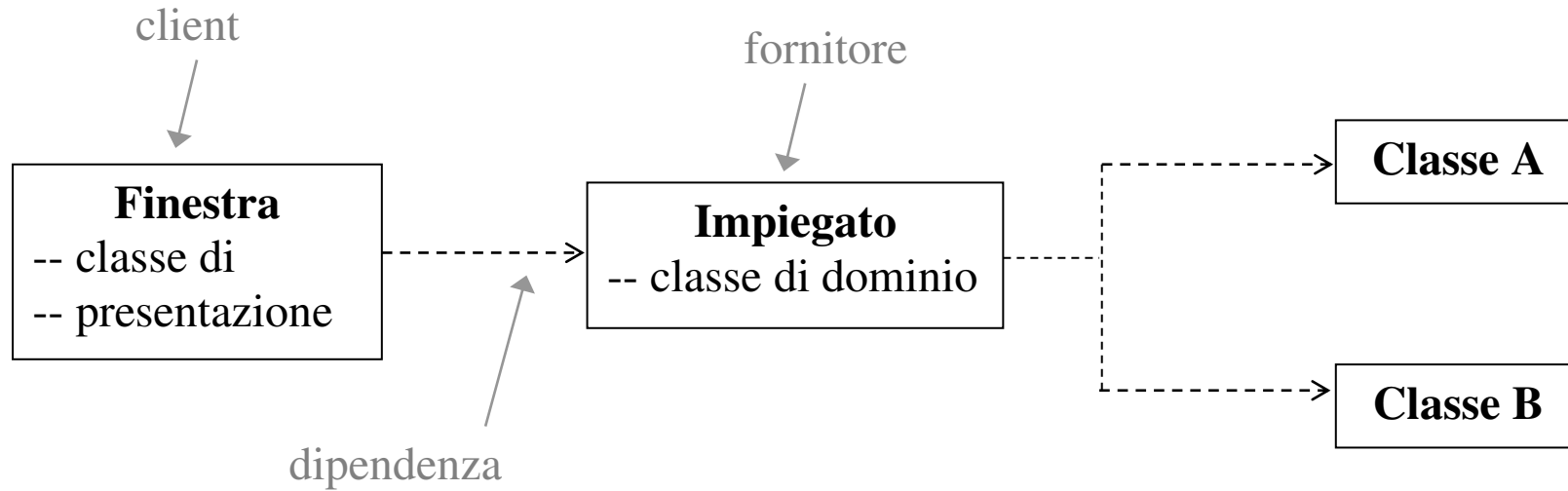
In generale non è una relazione transitiva (ad es. se C1 dipende da C2 che dipende da C3, la relazione non è transitiva se il cambiamento di C2 – conseguente a un qualsiasi cambiamento di C3 – riguarda solo il corpo delle operazioni di C2)

Molte relazioni UML implicano una dipendenza, ad es.

- in un'associazione navigabile $C1 \rightarrow C2$, C1 dipende da C2
- una sottoclasse dipende dalla superclasse (ma non viceversa)

Dato il codice di un sistema, le dipendenze in esso contenute sono rilevabili automaticamente da uno strumento CASE

Dipendenza e progettazione



Regole di progettazione

Separare presentazione (cioè le classi relative all'interfaccia) dalla logica del dominio (cioè le classi che incarnano il comportamento fondamentale del sistema), con la prima dipendente dalla seconda ma non viceversa

Minimizzare le dipendenze

Evitare i cicli di dipendenze, possibilmente fra classi dello stesso package e assolutamente fra classi di package diversi

Concetti avanzati

Elementi	Sintassi	Semantica
Parola chiave	Parola racchiusa fra virgolette uncinata Es. «interface»	Rappresenta un costrutto creato dall'utente perché non compreso in UML ma è simile a un altro che vi è compreso

Parola chiave \neq stereotipo

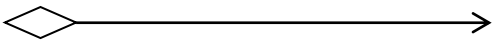
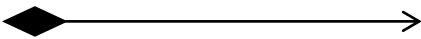
Stereotipo = meccanismo di estensione di UML che agisce sul metamodello

Profilo = estensione di una parte di UML ottenuta per mezzo di un gruppo coerente di stereotipi, adattando così il linguaggio a un particolare dominio (ad es. la modellazione di business)

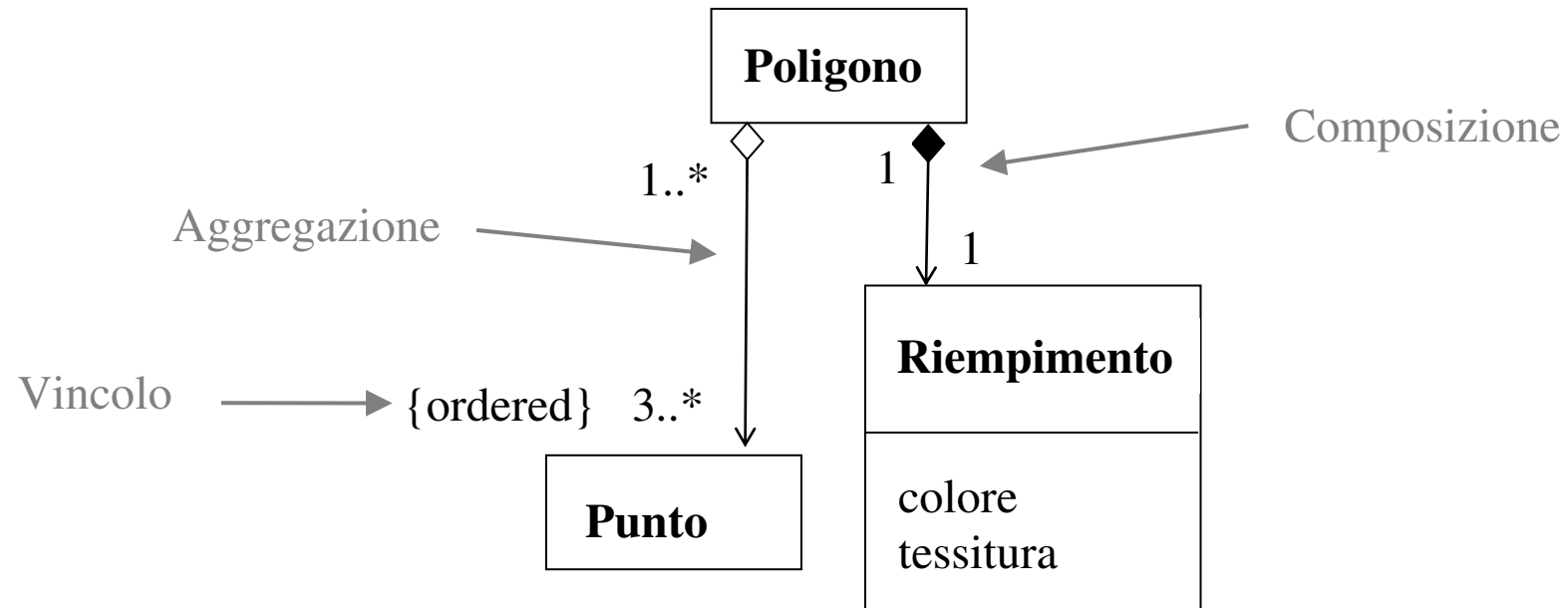
Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Operazione e attributo statici	Come operazione e attributo d'istanza ma sottolineati	Metodi e attributi con campo d'azione di classe (anziché d'istanza)

Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Aggregazione	<p>Freccia che termina sulla classe “parte” con una punta biforcuta; all’altra estremità si diparte dalla classe “tutto” con un rombo vuoto; su entrambe le estremità è obbligatoria la molteplicità</p> 	<p>Generica relazione “parte-di” (o HAS-A), difficilmente distinguibile dall’associazione</p> <p>Inclusa in UML senza definirne la semantica → persone diverse la usano con significati diversi</p>
Composizione	<p>Come nell’aggregazione ma il rombo è pieno</p>  <p>La molteplicità sul lato della classe composta è necessariamente 0..1 oppure 1</p>	<ul style="list-style-type: none"> • Varietà più forte dell’aggregazione: l’oggetto componente può appartenere a un solo oggetto composto (<u>regola di non condivisione</u>) • la cancellazione dell’oggetto composto si propaga a tutti gli oggetti componenti

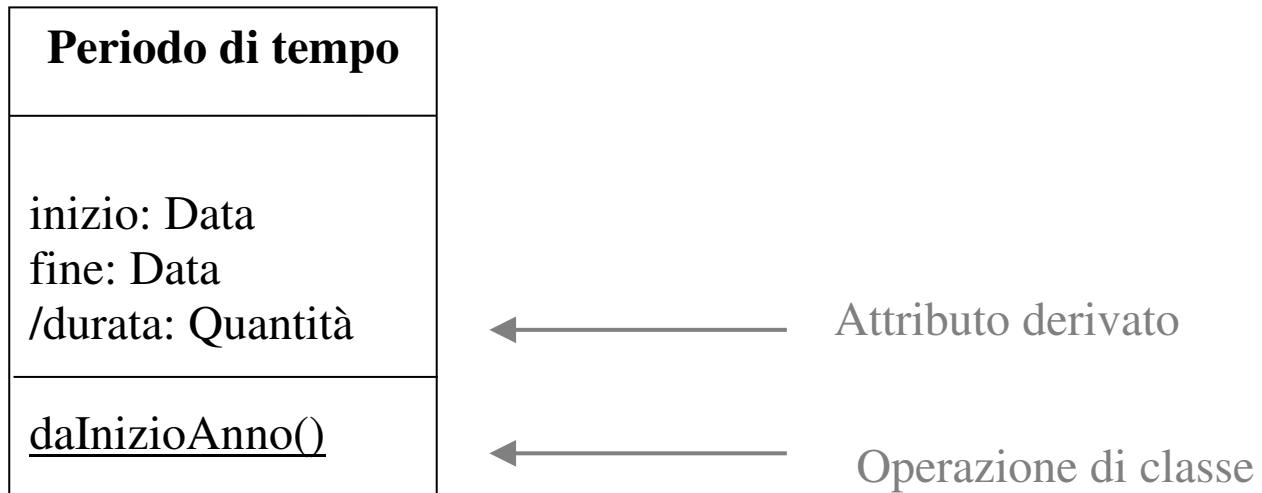
Aggregazione e composizione



Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Proprietà derivata	Il nome dell'associazione o dell'attributo è preceduto da una barra (/)	<ul style="list-style-type: none">• È un attributo/associazione rappresentato esplicitamente anche se può essere derivato a partire da altri attributi/associazioni• dal punto di vista concettuale, indica un vincolo tra valori e può essere usato per ricordarsi di chiedere conferma agli esperti circa una presunta derivabilità• dal punto di vista della specifica, indica un vincolo tra valori (non cosa viene direttamente memorizzato e cosa viene derivato)• dal punto di vista implementativo indica i campi usati come cache per motivi di efficienza

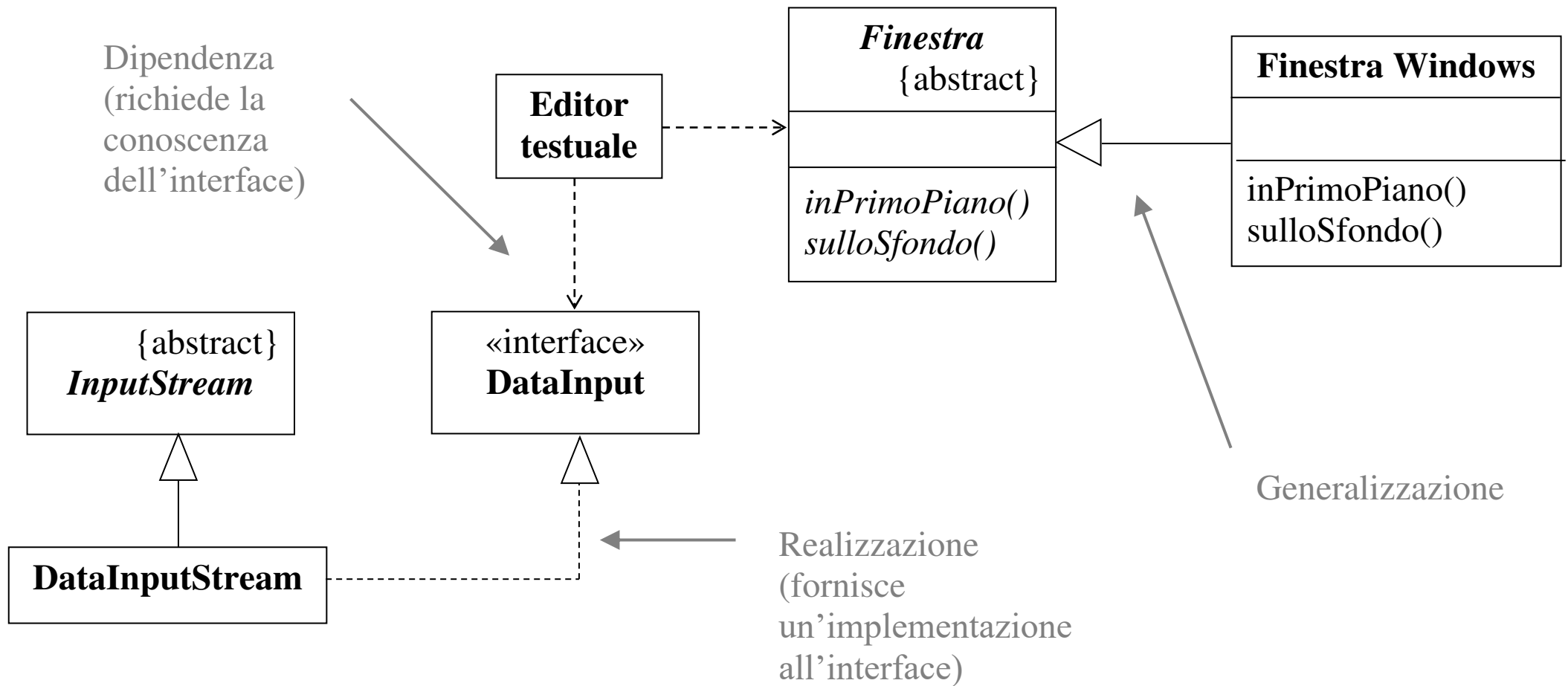
Concetti avanzati (cont.)



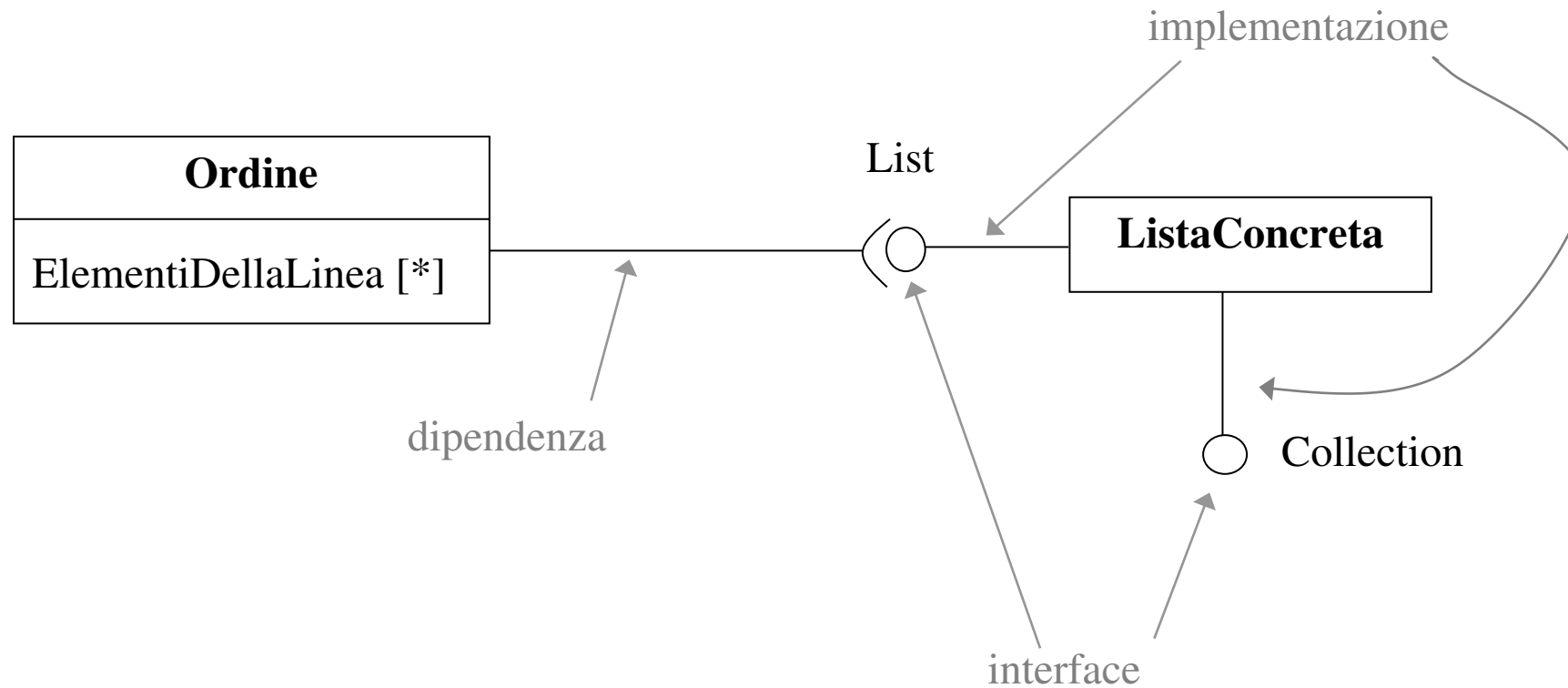
Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Classe e operazione astratti	Nome in corsivo + eventuale <i>vincolo</i> {abstract}	Operazione astratta = operazione priva di implementazione Classe astratta = classe non istanziabile direttamente
Interface	Uso della parola chiave «interface» entro la scatola della classe	Classe priva di implementazione
Realizzazione (o implementazione)	Freccia con linea tratteggiata e punta vuota diretta dalla classe all'interface	<ul style="list-style-type: none">• Relazione fra una classe (concreta o astratta) che implementa un'interface e l'interface stessa• a livello di specifica, realizzazione e subtyping sono indistinguibili

Classi astratte e interface



Interface: rappresentazione compatta



Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Associazione qualificata	Dal riquadro della classe sorgente fuoriesce un riquadro più piccolo, contenente il nome del qualificatore, da cui si diparte la linea continua che rappresenta l'associazione; tale linea termina con una punta che indica la navigabilità	<p>A ogni istanza della classe sorgente corrispondono gli elementi di un array associativo/tabella hash/mappa/dizionario indicizzati dal valore di una chiave (il qualificatore)</p> <p>La molteplicità dell'associazione qualificata va considerata nel contesto del qualificatore (ovvero indica quante istanze della classe destinazione corrispondono a un singolo valore/istanza del qualificatore)</p> <p>Nella modellazione concettuale rappresenta un vincolo</p>

Associazione qualificata



Classificazione

Classificazione = relazione tra un oggetto e il suo tipo; può essere

- singola: un oggetto appartiene a un solo tipo
- multipla: un oggetto può essere descritto da più tipi

Classificazione multipla \neq ereditarietà multipla

Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Classificazione multipla (subtyping)	<ul style="list-style-type: none">• Come la generalizzazione, ma etichettata con il nome di un <i>insieme di generalizzazione</i> (discriminante), che indica la base della classificazione;• il discriminante è accompagnato dal <i>vincolo</i> {complete} se ogni istanza della superclasse deve essere anche un'istanza di uno dei sottotipi del gruppo così contraddistinto;• ogni combinazione fra sottotipi aventi discriminante distinto deve essere legale;• tutti i sottotipi aventi lo stesso discriminante (gruppo di sottotipi) devono essere disgiunti	<ul style="list-style-type: none">• Associa a una classe (la superclasse) più sottotipi ortogonali (le sottoclassi)• utile a livello di modellazione concettuale

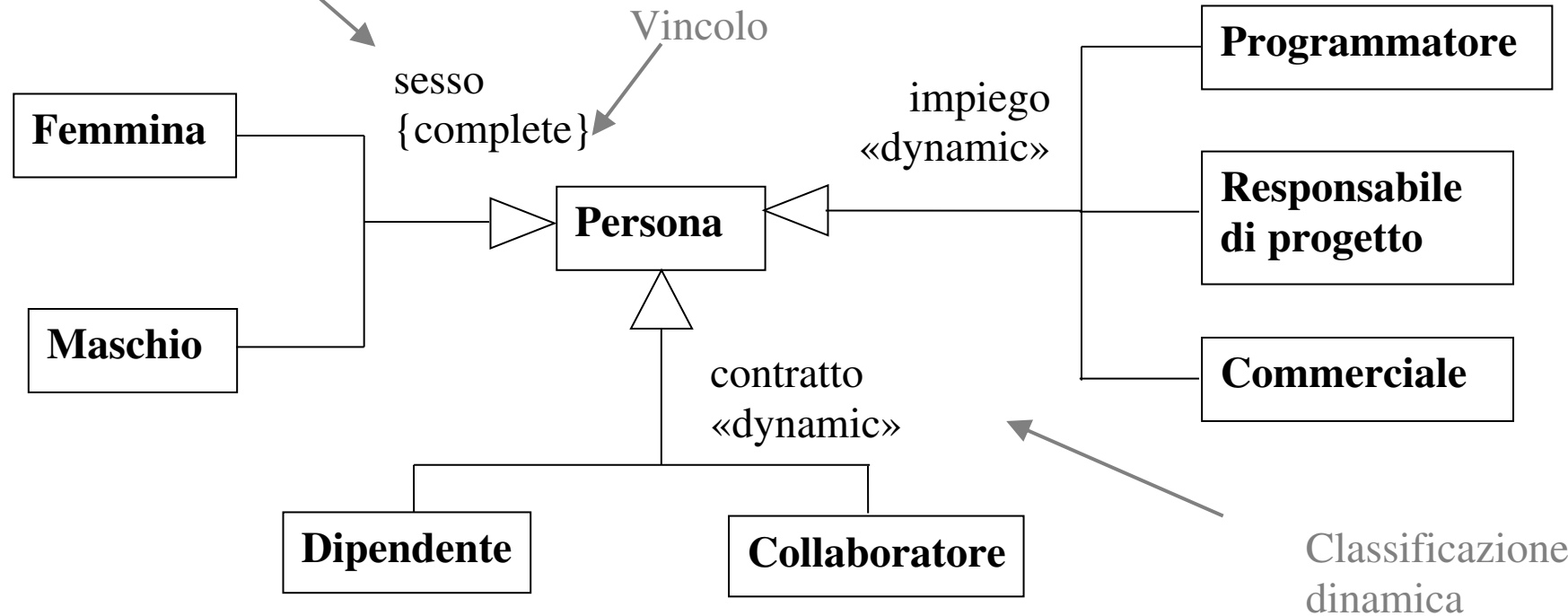
Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Classificazione dinamica	Come la classificazione multipla ma l'insieme di generalizzazione è accompagnato dalla parola chiave «dynamic»	<ul style="list-style-type: none">• Consente agli oggetti di cambiare tipo all'interno di una struttura di sottotipi• utile a livello di modellazione concettuale

Suggerimento: usare sempre una classificazione singola e statica (che corrisponde all'uso di un singolo anonimo insieme di generalizzazione)

Classificazione

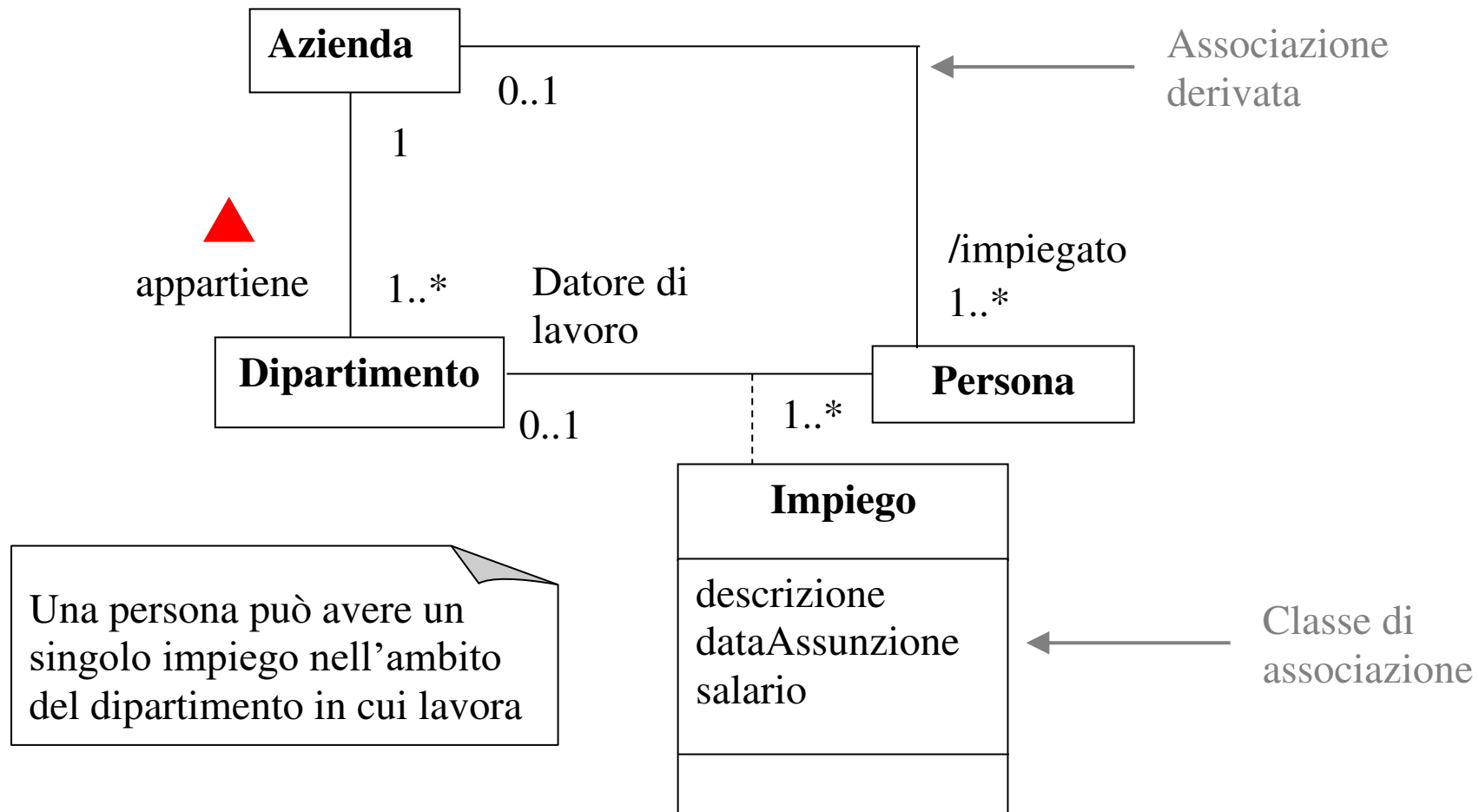
Insieme di
generalizzazione
(discriminante)



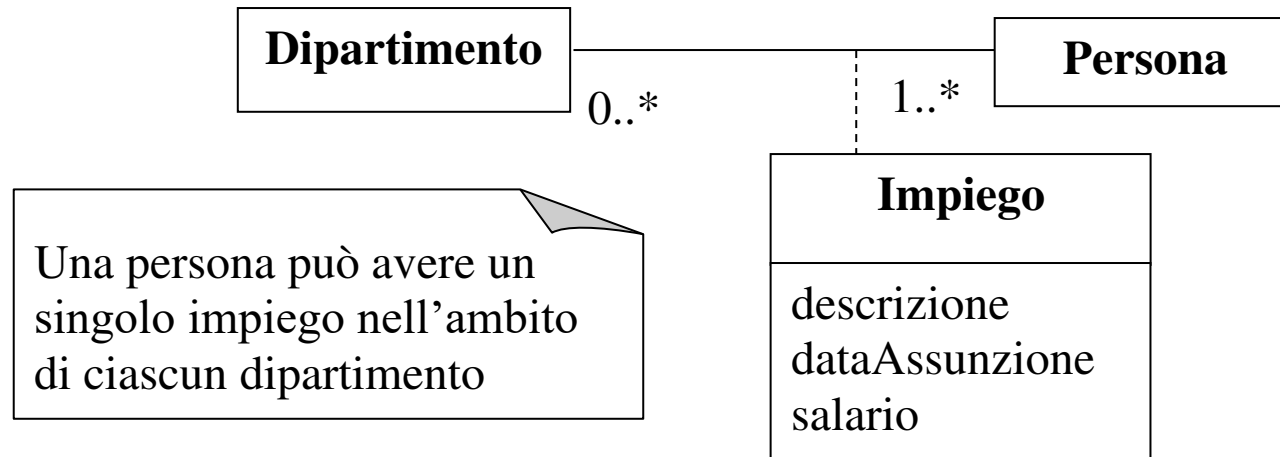
Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Classe di associazione	Classe collegata a una linea di associazione mediante una linea tratteggiata	Aggiunge un vincolo extra: ci può essere solo un'istanza della classe di associazione tra ogni coppia di oggetti associati
Classe parametrica (o template)	Classe con un riquadro tratteggiato sull'angolo superiore destro, contenente i parametri (tipi di dati) della classe, che possono essere più d'uno	<ul style="list-style-type: none">• Usata per lo più per definire collezioni• Raramente utile in fase di modellazione concettuale; da usare in fase di specifica e implementazione solo se effettivamente supportata dal linguaggio di programmazione (è inclusa con il nome di <i>classe generica</i> in Java e C#)

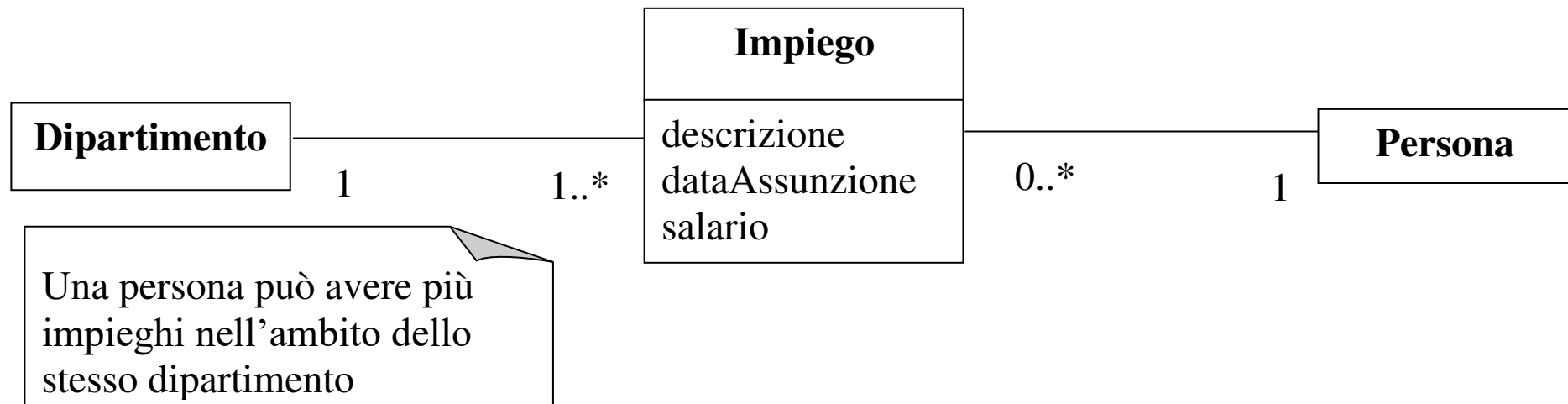
Classe di associazione



Promozione di una classe di associazione



Ma il significato è diverso rispetto a quello del diagramma sottostante, usato soprattutto per rappresentare relazioni temporali (storiche)



Concetti avanzati (cont.)

Elementi	Sintassi	Semantica
Derivazione	<ul style="list-style-type: none">• Dalla classe derivata si diparte una freccia di generalizzazione diretta alla classe template; la freccia è affiancata dalla parola chiave «bind» seguita dall'elenco dei nomi dei tipi usati come parametri fra parentesi angolari (<>), ciascuno preceduto da <i>nome_parametro</i> :: Es. <T::Dipendente>, oppure• La classe derivata contiene il nome della classe template, la parola chiave «bind» seguita dall'elenco dei nomi dei tipi usati, nello stesso formato di cui al punto precedente	<p>Una classe derivata rappresenta un uso particolare della classe parametrica, corrispondente a una specifica configurazione dei valori dei parametri</p> <p>La classe derivata non è una sottoclasse: infatti non è lecito aggiungerle caratteristiche rispetto a quelle del template</p>

Classi parametriche (template)

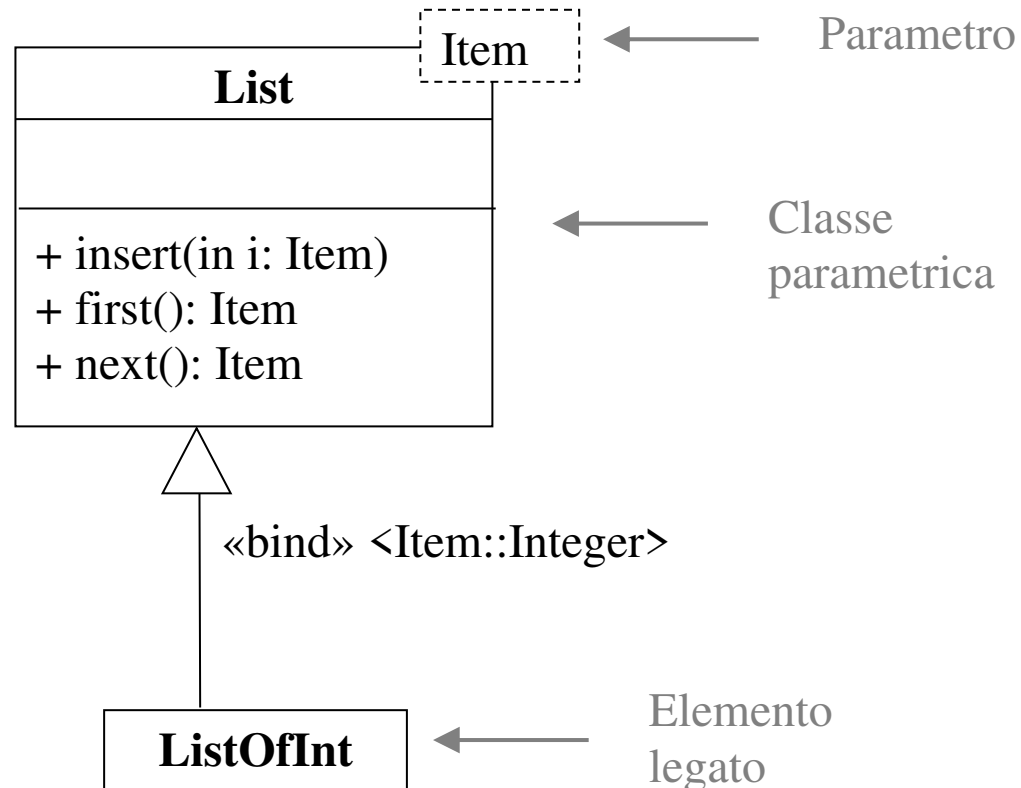
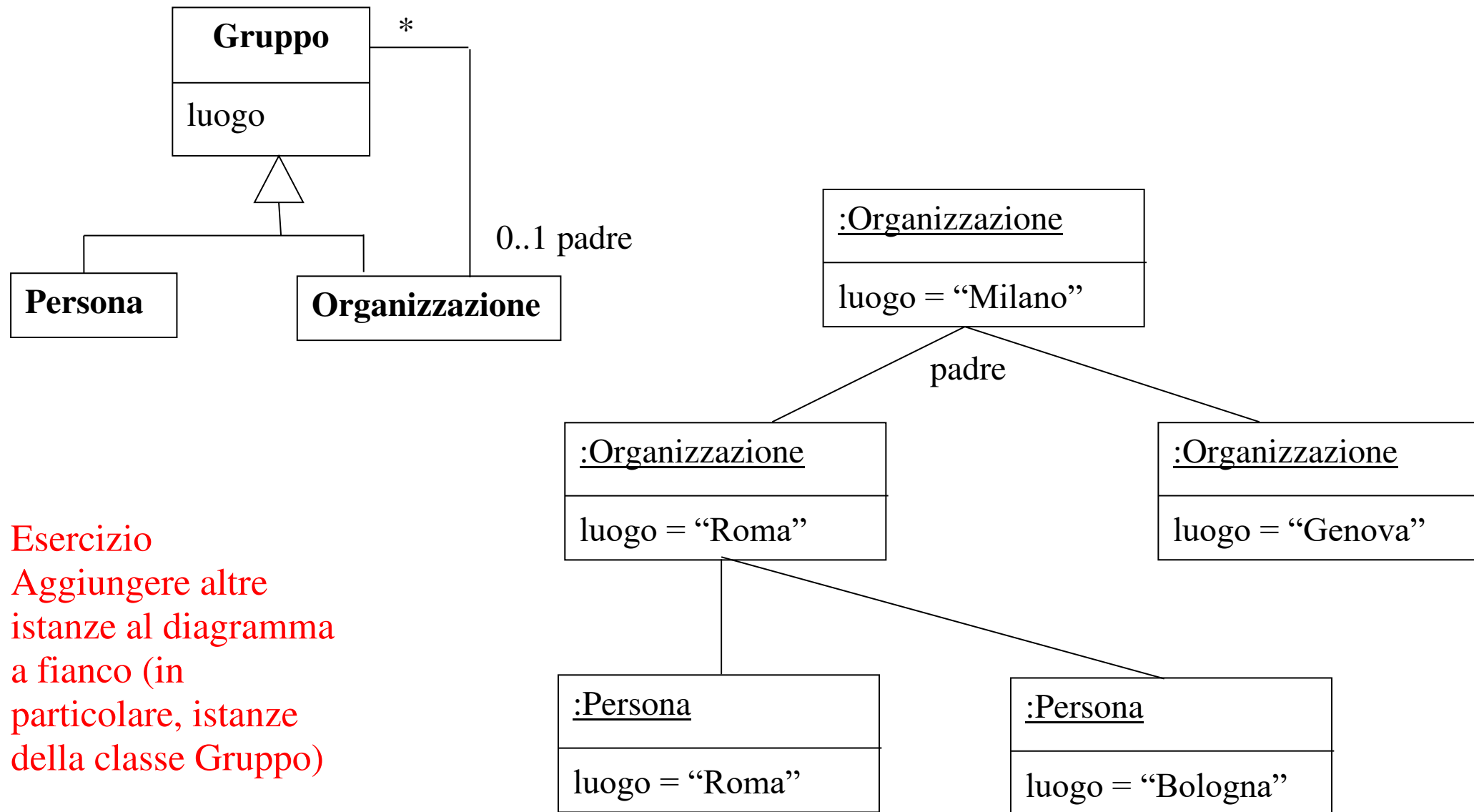


Diagramma degli oggetti (o delle istanze)

- È la fotografia degli oggetti che compongono un sistema sw in un dato momento
- È utile quando le connessioni fra oggetti sono complicate

Elementi	Sintassi	Semantica
Oggetto	Scatola suddivisa in due parti, entrambe dal contenuto opzionale, la prima contenente <u>nome-istanza : nome-classe</u> (può comparire o solo <u>nome-istanza</u> o solo <u>: nome-classe</u>), la seconda zero, uno o più <i>nome-attributo</i> = “ <i>valore</i> ” (l’elenco degli attributi non deve essere necessariamente esaustivo)	Istanza di una classe
Collegamento	Linea continua fra due oggetti, eventualmente etichettata con <i>nome-associazione</i>	Istanza di relazione fra due classi

Diagramma degli oggetti (cont.)



Esercizio
Aggiungere altre
istanze al diagramma
a fianco (in
particolare, istanze
della classe Gruppo)

Design

Progettazione

- É il ponte fra la specifica dei requisiti e la codifica
- É la fase in cui si decide come passare da “che cosa” deve essere fatto a “come” deve essere fatto
- La sua uscita principale si chiama architettura (o progetto) del sw

Architettura = definizione della struttura statica del sistema sw in termini di componenti e loro reciproche interconnessioni

Componente = parte di un sistema che fornisce risorse e servizi computazionali; può essere costituita a sua volta da componenti

Nella progettazione OO una classe è il più piccolo componente implementabile

Architettura

- Interna: si riflette nella suddivisione delle funzioni tra i vari componenti sw
 - ❖ di alto livello (diagramma UML dei componenti)
 - ❖ dettagliata (diagramma UML delle classi)
- Esterna: si traduce nella suddivisione dei componenti tra le risorse hw interessate e nelle interfacce verso l'hw (diagramma UML di deployment)

Architettura esterna

- Monolitica (o stand-alone): il sistema viene eseguito su un solo computer, senza comunicare via rete con altre parti del sistema
- Distribuita: il sistema è costituito da più applicazioni, operanti su macchine distinte

Client/server: le applicazioni sono collaboranti e ciascuna di esse implementa o un client o un server. I server funzionano su macchine a prestazioni elevate che spesso gestiscono una base di dati o un file system oppure sono collegati a una periferica (es. stampante). I client funzionano su macchine a prestazioni limitate (es. PC) e sfruttano i servizi dei server → la soluzione è vantaggiosa (anche) dal punto di vista dell'efficienza

Peer-to-peer: tutte le applicazioni sono dotate di funzioni e responsabilità simili, ciascuna di esse offre servizi e usa quelli delle altre

Master/slave (primario/secondario): l'applicazione master (o primaria) mantiene il controllo su tutte le applicazioni slave (o secondarie). Uno slave può chiedere al master un servizio, il master decide se concederglielo → pericolo di starvation. Si adotta per coordinare la gestione di risorse scarse

Obiettivi della modularizzazione

o, meglio, dei principi che la guidano:

- dominare la complessità (la somma delle complessità delle singole parti deve essere minore della complessità originaria)
- ridurre i tempi di produzione (grazie alla distribuzione e parallelizzazione del lavoro)
- favorire il riuso sistematico
- migliorare i fattori di qualità del sw

Progettazione architettuale

Moduli

- Modulo = parte di un sistema sw che fornisce un insieme di servizi ad altri moduli
- Servizio = elemento computazionale che gli altri moduli possono usare

Interfacce

Un modulo consiste di:

- Corpo = implementazione e suoi segreti
- Interfaccia = insieme dei servizi esportabili, definisce un contratto fra il modulo e i moduli suoi utenti; i (**moduli**) utenti conoscono solo l'interfaccia di un modulo

Relazioni (tutte irreflessive)

- USA: un modulo usa i servizi esportati da un altro (**relazione UML di dipendenza**)
- È-UN-COMPONENTE-DEI: descrive l'aggregazione di moduli in moduli di livello più alto (**diagramma UML dei componenti**)
- EREDITA-DA: per sistemi OO (**relazione UML di generalizzazione**)

Principi di modularizzazione

Progetto per il cambiamento

- Anticipare i cambiamenti
- Non concentrarsi sulle esigenze attuali ma prevedere la loro evoluzione (prototipazione evolutiva)
- Pensare al programma come al membro di una famiglia

Cambiamenti: verosimiglianza

- Degli algoritmi (ad es. per ragioni di efficienza)
 - Es. da ordinamento per affioramento a ordinamento per selezione
- Delle strutture dati (sia per ragioni di efficienza, sia per cambiamenti dei requisiti)
 - Es. dati dell'utente: 17% dei costi di manutenzione
- Funzionali (cambiamenti dei requisiti)
 - Es. l'introduzione dell'applicazione genera nuovi bisogni e modifica quelli esistenti, oppure si scoprono errori nei requisiti
- Della macchina astratta sottostante
 - Es. periferiche hw, OS, DBMS, ecc. (→ nuovi rilasci, problemi di portabilità, ecc.)
- Dell'ambiente
 - Es. anno 2000, euro

Cambiamenti: rischi

Passare da un progetto ordinato, ben documentato, riusabile, con codice pulito a un progetto contorto, non riusabile, con documentazione inconsistente, con codice “spaghetti”

Cambiamenti: dove e come

- L'interfaccia di un modulo è un contratto con i **moduli** clienti e, come tale, deve essere stabile
- Se le parti modificabili sono quelle segrete del modulo, il loro cambiamento non deve avere effetti per i **moduli** clienti
- Minimizzare il flusso di informazioni di un modulo verso i **moduli** clienti
- Minimizzare la presenza della relazione USA

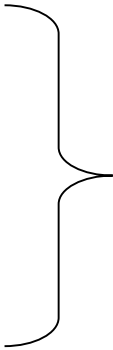
Cambiamenti: un esempio

Una TABELLA in cui è possibile inserire, cancellare, modificare e stampare voci (secondo un ordine prestabilito)

Interfaccia: INSERT, DELETE, MODIFY e PRINT

Si possono cambiare liberamente:

- Le strutture dati
- La politica di gestione dei dati (mantenere le voci ordinate oppure ordinarle solo prima di stamparle)
- Gli algoritmi (ad es. l'algoritmo di ricerca)



Questi sono
i segreti di
un modulo

Progettare rivolti all'interfaccia, non all'implementazione

Famiglia di programmi

Obiettivo del fatto di pensare al programma come al membro di una famiglia: progettare l'intera famiglia anziché ogni membro separatamente

Ad es. sistema di biglietteria

- Ferroviaria: formulazione interattiva del piano del viaggio (magari suddiviso in più tratte), scelta della classe e del posto da parte dell'utente, calcolo del prezzo, eventuale acquisto on-line, eventuale prenotazione, scelta fra modalità di pagamento diverse, invio messaggio di posta elettronica
- Marittima: molte funzionalità sono simili ma alcune diverse (ad es. possibilità di portare l'auto al seguito)

Principi di modularizzazione

Occultamento delle informazioni (**information hiding**, Parnas 1972): “define what you wish to hide and design a module around it”

- Un modulo è un'unità logica autocontenuta
- Entro questa unità è necessario distinguere fra ciò che un modulo fa per gli altri e come lo fa (i suoi segreti)
- Il modulo deve essere un firewall intorno ai suoi segreti
- I segreti devono essere incapsulati e protetti, ovvero l'accesso agli stessi deve essere filtrato dall'interfaccia

Altri principi e concetti chiave della modularizzazione

- Massima coesione e minimo accoppiamento
- Progetto per il riuso
- Astrazione
- Estensibilità
- Strutturazione a macchine virtuali

Astrazione

Tipo di modulo	Equivalente
Operazione astratta	Procedure della programmazione procedurale
Oggetto astratto (ad es. il modulo TABELLA) <ul style="list-style-type: none">– Un modulo che incapsula una struttura dati– Esporta un insieme di operazioni– L'applicazione delle operazioni modifica lo stato dell'oggetto astratto	
Tipo di dato astratto <ul style="list-style-type: none">– Un modulo che consente l'istanziamento di oggetti astratti	Classe della programmazione OO (+ classi astratte + interface)

Circa la relazione USA

Utilizzare solo una gerarchia di USA, non un grafo ciclico, perché è più facile

- da comprendere
- da sviluppare e verificare (se non è una gerarchia, si sviluppa un sistema in cui non funziona niente fino a quando non funziona tutto)

I livelli della gerarchia diventano i livelli di astrazione del sistema

Classi progettuali

Derivano da tre fonti:

- dominio dell'applicazione
- infrastruttura
- interfaccia utente

Sono:

- Classi dell'interfaccia utente (classi di presentazione)
- Classi del dominio (logica del dominio): sono frequentemente raffinamenti delle classi del modello concettuale del dominio
- Classi dei processi: rappresentano le astrazioni operative di basso livello necessarie per gestire al meglio le classi del dominio
- Classi persistenti: rappresentano archivi di dati
- Classi di sistema: consentono al sistema di comunicare con l'ambiente di calcolo e il mondo esterno

Progettazione per contratto (**Design by contract**)

- Tecnica sviluppata da Bertrand Meyer nel 1992 come caratteristica del linguaggio Eiffel
- Rifinisce un principio di progettazione noto, particolarmente adatto per la progettazione OO

Contratto = accordo fra un cliente e un contraente

Progettazione per contratto	Progettazione OO
Modulo contraente	Classe contraente
Contratto	Interfaccia della classe contraente
Modulo cliente	Classe cliente (che USA la classe contraente)

Un esempio di contratto nel mondo reale

	Obblighi	Benefici
Cliente	<ul style="list-style-type: none">◆ Fornire un terreno di dimensioni minime prefissate◆ Mettere a disposizione un punto di prelievo dall'impianto idraulico◆ Pagare una certa somma secondo modalità di versamento prefissate	Disporre di una piscina funzionante, di dimensioni e caratteristiche note, allacciata all'impianto idraulico
Contraente	<ul style="list-style-type: none">◆ Costruire una piscina funzionante, di dimensioni e caratteristiche note, effettuando l'allacciamento all'impianto idraulico◆ Nessun obbligo definito se gli impegni assunti dal cliente non sono rispettati	Ricevere, secondo le modalità prefissate, i versamenti della somma di denaro pattuita

Il contratto per un metodo OO

	Obblighi	Benefici
Metodo cliente	Precondizioni = ciò che è richiesto dal metodo contraente = condizioni sullo stato dell'oggetto su cui il metodo è invocato + condizioni sui parametri d'ingresso. Devono essere vere al momento dell'invocazione del metodo contraente	<ul style="list-style-type: none">♦ Il servizio computazionale reso dal modulo contraente♦ Conoscere le condizioni soddisfatte dai risultati, senza bisogno di controllarle
Metodo contraente (o servente)	Postcondizioni = ciò che deve essere fornito dal metodo contraente = condizioni sullo stato dell'oggetto su cui il metodo è stato invocato + condizioni su valore di ritorno e parametri d'uscita. Devono essere vere al termine dell'esecuzione del metodo contraente	Non dovere considerare tutte le possibili configurazioni d'ingresso (principio di progettazione)

Il contratto per un metodo OO

int intsqr(int i)

precondizioni: $i \geq 0$

postcondizioni: $r*r \leq i < (r+1)*(r+1)$, dove r è il valore di ritorno

int intsqr(int i)

precondizioni: true

postcondizioni: $(i < 0 \text{ AND } r = -1) \text{ OR } (i \geq 0 \text{ AND } r*r \leq i \text{ AND } i < (r+1)*(r+1))$

int intsqr(int i)

precondizioni: true

postcondizioni: $r*r \leq |i| \text{ AND } |i| < (r+1)*(r+1)$

Progettazione per contratto (cont.)

Usa tre tipi di asserzioni (cioè espressioni booleane che possono risultare false solo in presenza di errori di programmazione):

- precondizioni
 - postcondizioni
 - invarianti
- } Ciascuna precondizione o postcondizione è associata specificamente a un singolo metodo e, in generale, è diversa per ogni metodo
- Ciascun invariante è associato specificamente a una singola classe e, in generale, è diverso per ogni classe

Precondizioni

Se p è la precondizione del metodo servente m , o scriviamo nel codice del metodo cliente (per un qualsiasi oggetto x)

```
if x.p(par)
  then x.m(par)
  else ...{trattamento speciale}
```

o, ragionando sul programma, assicuriamo che p sia vera (per ogni oggetto x) prima della chiamata

Attenzione: per consentire ai clienti del metodo m contenuto nella classe K di verificare la sua precondizione p , è necessario che p sia espressa in termini di operazioni (solo query) esportabili di K

Precondizioni (cont.)

- Scelta delle precondizioni = decisione di progetto per cui non esistono regole guida
- Precondizioni deboli implicano che tutte le complicazioni sono delegate al metodo (→ difficoltà di sviluppo)

Postcondizioni

Servono a specificare cosa fa un'operazione senza dire come lo fa, cioè a separare l'interfaccia dall'implementazione

Invariante

Invariante = proprietà che (lo stato di) ogni istanza di una classe deve soddisfare

- dopo la sua creazione
- sia prima, sia dopo ogni operazione compiuta sull'istanza stessa (ma non necessariamente durante)

Rappresenta un ulteriore obbligo che la classe deve soddisfare con la sua implementazione

Può essere aggiunto (in congiunzione logica) alle precondizioni e postcondizioni di ciascuna operazione di una classe

Se tutti gli attributi di una classe sono privati, basta aggiungerlo (in congiunzione logica) alle postcondizioni del costruttore e dei metodi modificatori

Proprietà interne di una classe

Esempio: classe tabella (il contenuto di ciascuna istanza è un insieme di elementi, senza ripetizioni)

- Invariante: $n_elementi \leq \text{capacità}$
- Operazione *inserisci(elemento)*
 - Precondizione: $(n_elementi < \text{capacità})$ AND (*elemento* non è in tabella)
 - Postcondizioni:
 - ✓ *elemento* è presente nella tabella
 - ✓ $n_elementi' = n_elementi + 1$
 - ✓ tutti gli elementi che erano già presenti in tabella prima dell'inserimento lo sono ancora

Le asserzioni

- esplicitano le responsabilità circa i controlli da effettuare da parte dei moduli, evitando così ridondanza di controlli o controlli insufficienti
- risolvono i conflitti di responsabilità in presenza di errori

Eccezione

Si verifica a run time quando un'operazione (viene invocata nel rispetto della sua preconditione ma) non è in grado di terminare la propria esecuzione nel rispetto della postcondizione

N.B. in questa definizione sia preconditione che postcondizione sono comprensive dell'invariante della classe

Progettazione difensiva (**Defensive programming**)

Si anticipano i rischi di fallimento di un modulo **dovuti a mancato soddisfacimento delle asserzioni** e si decide se evitarli o tollerarli

Eccezione = evento attraverso cui il modulo servente, prima di terminare la sua esecuzione, segnala al modulo cliente che non è riuscito a completare correttamente il servizio richiesto (→ distinzione fra terminazione normale e terminazione anomala). Il modulo cliente risponde gestendo l'eccezione in modo appropriato

I tipi di eccezioni sollevate da un modulo (**clausola throws di Java**) fanno parte della sua interfaccia

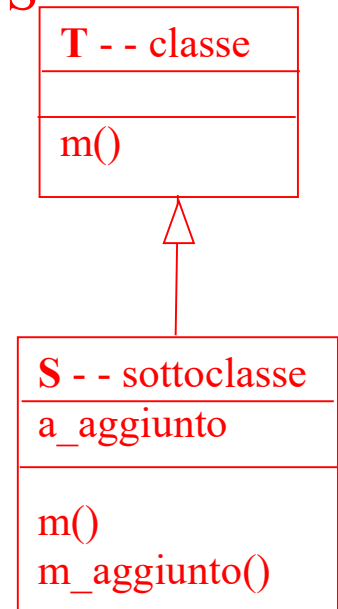
Le modalità di gestione delle eccezioni ricevute da un modulo fanno parte dei segreti di tale modulo

Asserzioni ed ereditarietà

Sostituibilità: se S è una sottoclasse di T , allora, durante l'esecuzione, il valore di una variabile o parametro v di tipo T può essere un oggetto istanza di S

Le sottoclassi possono

- Aggiungere attributi e metodi, rispettando però il vincolo semantico
 $\text{inv}_{\text{sottoclasse}} \rightarrow \text{inv}_{\text{classe}}$
- Ridefinire i metodi, soddisfacendo però i seguenti vincoli semantici:
 $\text{post}_{\text{sottoclasse}} \rightarrow \text{post}_{\text{classe}}$
 $\text{pre}_{\text{classe}} \rightarrow \text{pre}_{\text{sottoclasse}}$



In questo modo le asserzioni impediscono che le operazioni ridefinite o aggiunte nella sottoclasse siano inconsistenti con quelle della classe padre (**ovvero garantiscono che le “sostituzioni” avvengano senza alterare la correttezza dei risultati del programma**)

Uso ideale ed effettivo delle asserzioni

Ideale

Le asserzioni dovrebbero essere incluse sia nel progetto, sia nel codice come parte della definizione di interfaccia

Effettivo

- Le asserzioni vengono incluse nel codice per il testing e il debugging e vengono rimosse quando si compila il codice finale
- Il programma deve ignorare la presenza del controllo delle asserzioni
- L'uso delle sole precondizioni spesso consente di rilevare il massimo numero di errori, con il minimo spreco di risorse computazionali
- Le asserzioni sono parte integrante di Eiffel e anche di altri linguaggi, fra cui Java

Asserzioni in Java

Sintassi:

```
assert condizione // condizione booleana  
assert condizione : spiegazione // spiegazione è una stringa
```

Esempio.

```
assert count > 0 : "violated precondition size() > 0"
```

Semantica:

se *condizione* è vera l'esecuzione procede, altrimenti viene lanciata un'eccezione di tipo `AssertionError` la cui gestione visualizza un msg d'errore (nome_file, #linea, stringa *spiegazione*)

Assertzioni in Java (cont.)

Introdotte nella versione 1.4 di Java SDK

Il controllo delle asserzioni può essere abilitato e disabilitato (il meccanismo dipende dall'ambiente di esecuzione), anche in forma selettiva (ovvero limitatamente ad alcune classi/package)

Pre/postcondizioni e invarianti non esprimibili come asserzioni Java

Ipotesi: il primo indice di un array assume il valore 1

$\{n > 0\}$

procedure search (table: **in** integer_array; n: **in** integer;
 element: **in** integer; found: **out** Boolean);

$\{found \equiv (\text{exists } i (1 \leq i \leq n \text{ and } table(i) = element))\}$

$\{n > 0\}$

procedure reverse (a: **in out** integer_array; n: **in** integer);

$\{\text{for all } i (1 \leq i \leq n) \text{ implies } (a'(i) = a(n - i + 1))\}$

Per il controllo di queste postcondizioni è necessaria l'esecuzione di programmi ciclici

Java Modeling Language (JML)

www.cs.ucf.edu/~leavens/JML/index.shtml

La pagina web citata rimanda all'elenco aggiornato di tutti i tool scaricabili

Es. L'algoritmo di ricerca binaria effettua correttamente la ricerca solo se l'array di ingresso è ordinato in ordine non decrescente

```
/*@ requires vet != null
   @      && (\forall int i;
   @      0 < i && i < vet.length;
   @      vet[i - 1] <= vet[i])
   @*/

int binarySearch(int[] vet, int x) { ... }
```

Nota: l'uso di queste specifiche evita controlli difensivi inefficienti all'interno del codice Java

Programma Java contenente specifiche JML

- È traducibile da `jmlc`, estensione del compilatore Java SDK 1.4. Il byte code risultante include i controlli run-time di verifica delle condizioni espresse in JML (disabilitabili quando il programma va in produzione). L'esecuzione dei controlli delle asserzioni è trasparente, fatta eccezione per le prestazioni
- È interpretabile direttamente da `jmlrac`
- È verificabile da `jmlunit`, strumento di testing che estende `Junit` combinandolo col compilatore JML, che controlla se il comportamento del codice in corrispondenza dei casi di test soddisfa le condizioni espresse in JML (ovvero solleva i programmatori dalla scrittura di oracoli)
- È utilizzabile da `jmldoc`, strumento che genera documentazione esterna HTML comprendente sia i commenti Javadoc, sia le specifiche formali espresse in JML
- È analizzabile da `escjava2` (extended static checker), che può trovare difetti quali puntatori potenzialmente nulli o violazioni dei limiti di array. Esso sfrutta e propaga le specifiche JML

Istruzioni JML

Sono controllabili da `jml`, che sostituisce `jmlc` nel caso non sia necessario compilare il codice Java

Tutti i tool citati sono disponibili gratuitamente

È (stato) disponibile (ed è ritornato tale, si veda <https://www.pm.inf.ethz.ch/research/jml-plugin.html>) anche un plugin per l'IDE Eclipse che supporta la sintassi JML e si interfaccia a vari tool che fanno uso di annotazioni JML

Diagrammi dei package

Package

- Meccanismo di raggruppamento di più classi (riferito al momento della compilazione) in unità di livello più alto, al fine di dominare la complessità strutturale di un sistema sw
- Supportato dai linguaggi di programmazione (come *namespace* in C++ e .NET)

All'interno di un programma Java, per fare riferimento a due classi omonime contenute in package distinti è necessario precisarne il pathname (relativo rispetto al direttorio radice dei package), ad esempio `java.pack1.C` e `java.pack2.C`

Per inserire una classe in un package, si scrive nella prima riga del file che definisce tale classe `package nome_package;`

In mancanza di indicazioni, una classe Java viene posta nel package (anonimo) di default

Dipendenza

- Fra due elementi esiste una (relazione di) dipendenza se i cambiamenti apportati alla definizione dell'uno si possono potenzialmente ripercuotere sull'altro
- Idealmente solo i cambiamenti dell'interfaccia di una classe dovrebbero ripercuotersi sulle altre
- Il principio del progetto per il cambiamento richiede di minimizzare le dipendenze, cosicché gli effetti dei cambiamenti siano ridotti e il sistema si possa modificare con minore sforzo
- UML prevede molti tipi di dipendenza, ognuno con la propria semantica e parola chiave (ad es. «call» e «create»)

Suddivisione delle classi fra package diversi

- La dipendenza è il metodo euristico maggiormente usato per raggruppare le classi in package
- Un altro criterio è il riuso comune, secondo cui le classi di un package dovrebbero essere usate insieme
- Un terzo criterio è la chiusura comune, secondo cui le classi di un package dovrebbero condividere le cause di un eventuale cambiamento

Dipendenza fra classi e fra package

- Ogni package può raggruppare classi pubbliche e/o private e/o protette (*protected*)
- Fra due package esiste dipendenza se c'è dipendenza fra (almeno) una classe del primo e un'altra (necessariamente pubblica) del secondo
- Le dipendenze fra package non sono transitive
- Un flusso di dipendenze unidirezionale è generalmente indice di un sistema ben strutturato

In Java la visibilità di una classe (non annidata) può essere solo pubblica (*public*) o privata (è quella di una classe priva di modificatori di visibilità, che pertanto è visibile solo all'interno del package di appartenenza)

Interfaccia di un package

- È l'insieme dei metodi pubblici delle classi pubbliche del package
- Per ridurre tale insieme si può dare a tutte le classi del package visibilità privata, aggiungendo poi altre classi pubbliche, denominate facciate (mediante la parola chiave «`facade`»), per implementare il comportamento esterno
- Le facciate delegano l'esecuzione delle proprie operazioni alle classi nascoste
- Man mano che il n° di dipendenze entranti in un package aumenta, la sua interfaccia deve diventare sempre più stabile

Package in Java

Supponiamo di definire, in Java, la classe `D` entro il package `pD`. Sia `C` una classe pubblica contenuta nel package `pC`. Per rendere visibile `C` a `D`, si può utilizzare il pathname di `C` (anziché il solo nome `C`, ad esempio, `java.pC.C`) entro il codice di `D` oppure importare `C` entro `D` (`import java.pC.C`) e poi usare solo il nome `C`

L'istruzione Java `import` precede la definizione di una classe (e segue l'indicazione `package nome`). Lo stesso file può contenere più `import`
Es. `import nome_package.nome_classe, import nome_package.*`

Classi e interface contenute nel package `java.lang` (ad es. `Object`, `String`, `System`, `Math`) sono tutte importate automaticamente in qualsiasi classe

È possibile indicare (in una apposita variabile d'ambiente) uno o più direttori radice dei package

Diagramma dei package

- Mostra più package di classi con le loro dipendenze
- Aiuta a individuare le dipendenze (al fine di ridurle)
- È uno strumento fondamentale per mantenere il controllo sulla struttura globale del sistema sw
- Nella prospettiva software, può essere generato a partire dal codice stesso

Diagramma dei package (cont.)

Elementi	Sintassi	Semantica
Package	<p>Scatola con linguetta (<i>tab</i>) in alto a sx. La linguetta può:</p> <ul style="list-style-type: none"> ▪ Essere vuota, nel qual caso il nome del package (in grassetto) è contenuto nella scatola ▪ Contenere il nome del package (in grassetto), nel qual caso la scatola contiene un elenco di classi, oppure un diagramma dei package e/o delle classi ▪ Contenere la parola chiave «<i>global</i>», che significa che tutti (o quasi) i package del sistema sono dipendenti dal package considerato <p>Ogni classe può essere dotata dell'indicatore di visibilità</p> <p>Il <u>nome</u> di ciascun package/classe può essere semplice o <u>completamente qualificato</u>; quest'ultimo mostra la struttura dei package, dal più esterno al più interno, che contengono l'elemento corrente, fino a tale elemento incluso, dove due nomi consecutivi sono separati da ::</p> <p>Es. <code>System::Data</code></p>	<p>L'aggregazione di più package entro un package consente la modellazione gerarchica di sistemi complessi</p> <p>Ogni classe fa parte di un solo package</p> <p>Ogni classe deve aver un nome distinto all'interno del package che la racchiude</p>

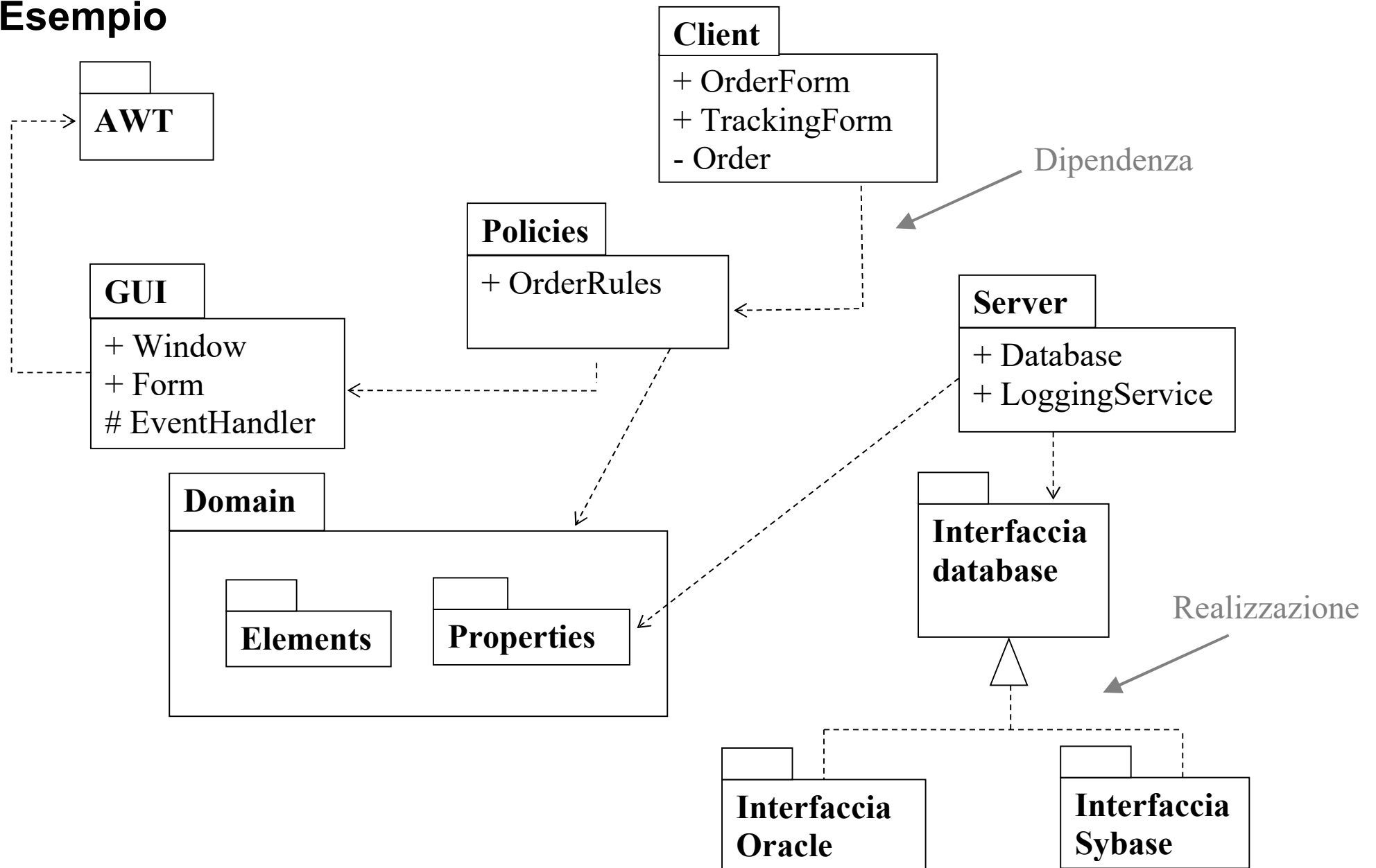
Diagramma dei package (cont.)

Elementi	Sintassi	Semantica
Dipendenza	Freccia con linea tratteggiata e punta biforcuta, uscente dal dipendente e terminante sul riquadro di un altro package o su quello di un elemento in esso contenuto	Una dipendenza verso un package che a sua volta ne contiene altri è la somma di più dipendenze di livello inferiore, cioè denota che ci sono dipendenze verso qualche elemento interno
Realizzazione (o implementazione)	Freccia, con linea tratteggiata e punta triangolare vuota, uscente dal package più specifico e terminante sul riquadro di un altro package o su quello di un elemento in esso contenuto	Il package più generale contiene (non esclusivamente) interface e classi astratte che sono implementate dal package più specifico

Suggerimenti

- Interpretare gli indicatori di visibilità delle classi in base alle regole del linguaggio di programmazione adottato dal sistema
- Minimizzare i cicli nella struttura delle dipendenze
- Se ci sono dei cicli, cercare di contenerli in un package più grande
- Sforzarsi di eliminare cicli nelle dipendenze tra i package del dominio e le interfacce esterne
- Generare automaticamente il diagramma dei package di un sistema già implementato come punto di partenza per migliorare la struttura del sistema attraverso una riduzione delle dipendenze (refactoring)
- Considerare il package come unità di base per il testing (può essere utile aggiungere delle classi che servono solo in supporto al testing delle altre classi nel package)

Esempio



Ingegneria dei requisiti

Requisiti

Proprietà (funzionali e non) che l'applicazione dovrà avere, descrivono

CHE COSA il sistema dovrà fare piuttosto che COME lo dovrà fare,

sono focalizzati sul

PROBLEMA non sulla SOLUZIONE

Requirements engineering

The process of establishing requirements, i.e. the SERVICES that the customer requires from a system and the CONSTRAINTS under which it operates and is developed

Types of requirement

- User requirements
 - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers
- System requirements
 - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor
- Software specification
 - A detailed software description which can serve as a basis for a design or implementation. Written for developers

Specifica è un termine ampio che significa “definizione”

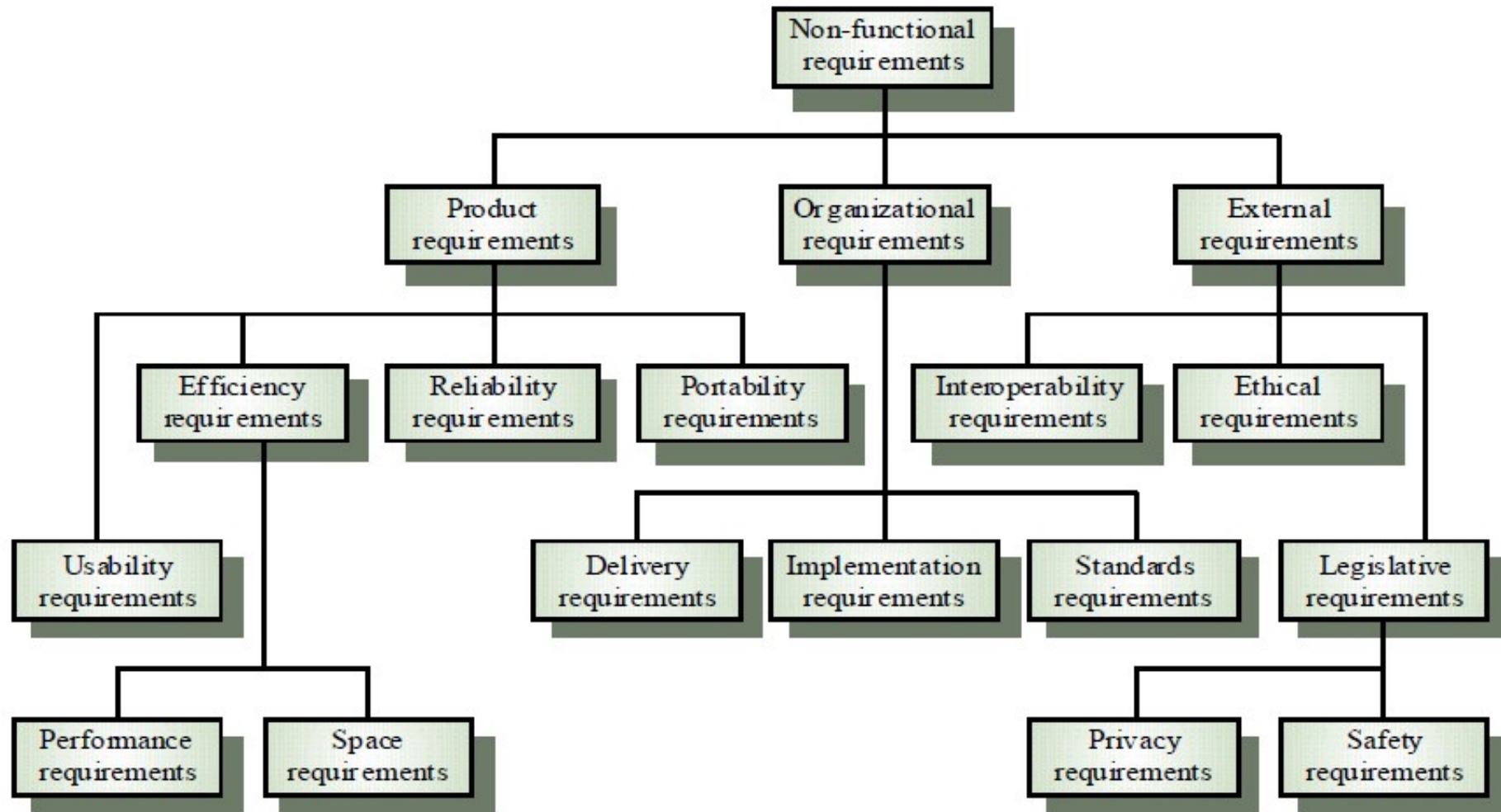
Functional and non-functional requirements

- Functional requirements
 - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Non-functional requirements
 - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

Non-functional requirements

- Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular CASE system, programming language or development method
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

Non-functional requirement types



©Ian Sommerville 2000

Software Engineering, 6th edition. Chapter 5

Slide 14

Elicitazione e analisi dei requisiti

- L'elicitazione dei requisiti è una attività critica per la riuscita del progetto: errori introdotti in questa fase hanno un costo enorme se scoperti troppo tardi
- I requisiti sono uno strumento molto importante di comunicazione con il committente
- L'analisi ha luogo quando l'esperto del dominio è presente, altrimenti è pseudoanalisi
- Uno degli elementi più importanti quando si gestiscono i rischi legati ai requisiti è avere accesso alla conoscenza degli esperti del dominio
- La mancanza di contatto con gli esperti è una delle cause più comuni di fallimento di un progetto

Problems of requirements analysis

- Stakeholders don't know what they really want
- Stakeholders express requirements in their own terms
- Different stakeholders may have conflicting requirements
- Organisational and political factors may influence the system requirements
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change

Caratteristiche dei requisiti

1. Validità – ogni requisito esprime qualcosa di cui l'utente ha realmente bisogno
2. Correttezza – la descrizione di ciascun requisito non contiene errori
3. Consistenza – non ci sono conflitti tra i requisiti
4. Completezza (esterna e interna) – tutti gli stati, le funzionalità, gli input, gli output e i vincoli sono contemplati da qualche requisito
5. Realismo – effettiva realizzabilità
6. Comprensibilità e univocità di interpretazione
7. Tracciabilità – riconducibilità alle ragioni / origini
8. Modificabilità
9. Verificabilità – quando si formula un requisito, si deve stabilire come decidere a posteriori se esso è stato realizzato dal sistema oppure no

Completezza dei requisiti

- Interna

Ogni nuovo concetto o vocabolo utilizzato deve essere definito (ad es. mediante un glossario)

- Esterna

La specifica deve documentare tutti i requisiti necessari
Difficoltà: quando ci si ferma?

Requisiti disambigui, comprensibili, precisi

Esempio (da non imitare) di un frammento di specifica di un word-processor

La selezione è il processo attraverso cui si designano aree del documento su cui si vuole lavorare. La maggior parte delle azioni di editazione e formattazione richiedono due passi: prima si seleziona ciò su cui si vuole lavorare, come testo o grafici, quindi si inizia l'azione appropriata.

Un altro esempio (da non imitare), relativo un sistema in tempo reale safety-critical

Il messaggio deve essere triplicato. Le tre copie devono essere inoltrate attraverso tre diversi canali fisici. Il ricevente accetta il messaggio sulla base di una politica di voto due-su-tre.

Requisiti logicamente consistenti

Esempio (da non imitare) di un frammento di specifica di un word-processor

L'intero testo dovrebbe essere mantenuto in linee di uguale lunghezza. La lunghezza è specificata dall'utente.
A meno che l'utente non dia un comando esplicito di sillabazione, un ritorno a capo dovrebbe avvenire solo alla fine di una parola.

INTERNATIONAL
STANDARD

ISO/IEC/
IEEE
29148

Second edition
2018-11

**Systems and software engineering —
Life cycle processes — Requirements
engineering**

Confermato nel 2024

Specifica dei requisiti

Tipologie di applicazioni (o di parti di esse)	Requisiti da specificare
Sequenziali (unico flusso di controllo)	Funzionalità offerte
Concorrenti (più flussi paralleli di controllo + meccanismi di sincronizzazione per l'accesso a risorse condivise)	Attività parallele, risorse condivise, meccanismi di sincronizzazione
Dipendenti dal tempo (dette “in tempo reale”: la correttezza dei risultati dipende dal tempo di esecuzione delle attività)	Tempi di esecuzione
Non dipendenti dal tempo	

Linguaggi di specifica

NON esiste un linguaggio di specifica adatto per qualunque problema o classe di applicazioni

Ulteriore categorizzazione delle applicazioni (o di parti di esse)	Requisiti da specificare	Esempi di linguaggi di specifica
Orientate ai dati (es. sistemi informativi)	Struttura concettuale dei dati	Modello ER
Orientate alle funzioni (es. traduttori)	Funzioni e flusso dei dati	Diagrammi di flusso (DFD)
Orientate al controllo (es. applicazioni dipendenti dal tempo che interagiscono con l'ambiente esterno)	Attività parallele, risorse condivise, tempi di esecuzione	Automi a stati finiti, reti di Petri, diagrammi di stato UML (statechart), diagrammi di attività

Linguaggi di specifica: classificazione

Specifiche	Esempi di linguaggi di specifica	Note
<i>Informali</i>	linguaggio naturale (che è impreciso, ambiguo e ridondante)	
<i>Formali</i> : in formalismo matematico (che obbliga alla precisione, è passibile di manipolazione automatica, talvolta consente di provare automaticamente consistenza e completezza, può essere eseguito)	Z (sequenziale), Reti di Petri (concorrente), FSM	Una specifica eseguibile costituisce un prototipo → può essere convalidata da committenti/utenti
<i>Semiformali</i> : notazioni (più o meno) rigorose, spesso grafiche + annotazioni in linguaggio naturale	ER, DFD, casi d'uso, diagrammi UML	

Linguaggi di specifica

Notazione =

sintassi del linguaggio di modellazione (aspetto grafico nel caso di linguaggi grafici; la sintassi UML è descritta da un meta-modello in UML)

Notazioni semiformali:

il loro significato non è univoco, bensì viene lasciato all'interpretazione dell'utente o dello strumento CASE generatore di codice (es. Together)

Linguaggi di specifica: ulteriore classificazione

- Operazionali
Specifica del comportamento desiderato attraverso una macchina astratta
- Descrittivi/dichiarativi
Specifica del comportamento desiderato attraverso le proprietà dello stesso

Specifiche operazionali e descrittive: un esempio

Specifica di una figura geometrica E

- Operazionale

E può essere disegnata come segue:

- 1) Selezionare due punti su un foglio solidale a un piano rigido
- 2) Inserire due perni in tali punti
- 3) Prendere una cordicella e fissarne le estremità ai perni
- 4) Tendere la cordicella mediante una matita, appoggiando la punta della matita sul foglio, segnando così il punto P
- 5) Muovere la matita in senso orario, mantenendo la punta sul foglio e la cordicella tesa, fino a raggiungere il punto P

- Descrittiva

$$a x^2 + b y^2 + c = 0$$

dove a, b e c sono costanti appropriate

Specifiche operazionali e descrittive: un altro esempio

- Operazionale

Sia a una sequenza di n elementi. Il risultato del suo ordinamento è una sequenza b di n elementi tale che il primo elemento di b è il minimo di a (se più elementi hanno lo stesso valore, ognuno di essi è accettabile); il secondo elemento di b è il minimo degli $(n - 1)$ elementi ottenuti togliendo da a il suo elemento minimo, e così via, finché tutti gli n elementi sono stati tolti da a .

- Descrittiva

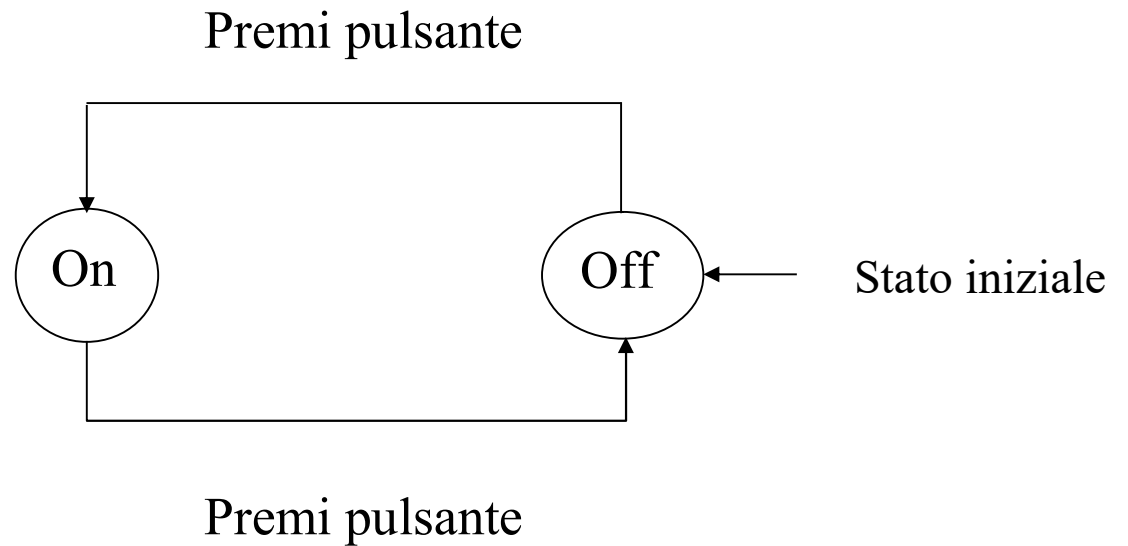
Il risultato dell'ordinamento di una sequenza a è una sequenza b che costituisce una permutazione di a tale per cui ogni elemento assume un valore minore o uguale di quello dell'elemento successivo.

Macchine a stati finiti (FSM)

S: insieme finito **non vuoto** di stati

I: insieme finito degli ingressi

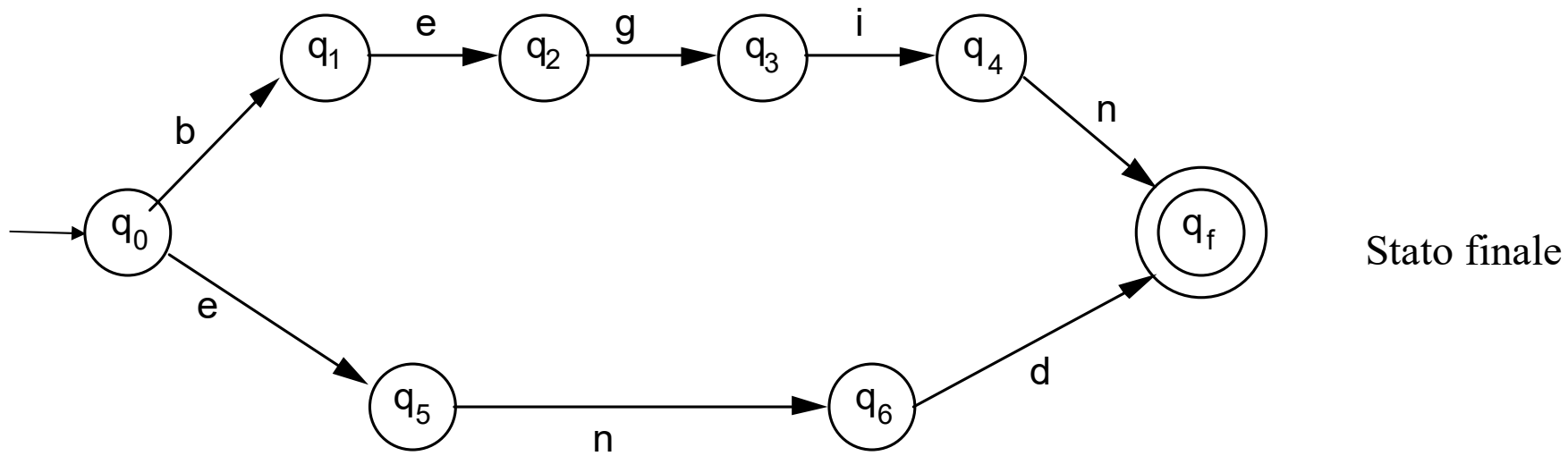
δ : funzione di transizione
(anche parziale)



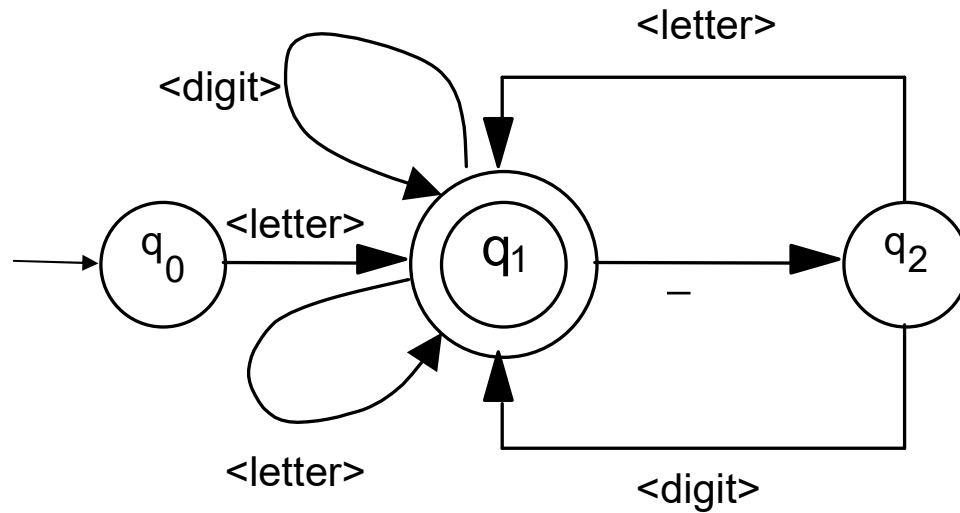
Classi di FSM

- Deterministiche ($\delta: S \times I \rightarrow S$) / nondeterministiche ($\delta: S \times I \rightarrow 2^S$)
- Riconoscitori (dotati di stati finali)
- Traduttori (dotati di un insieme finito di uscite)

FSM come riconoscitori



FSM come riconoscitori (cont.)

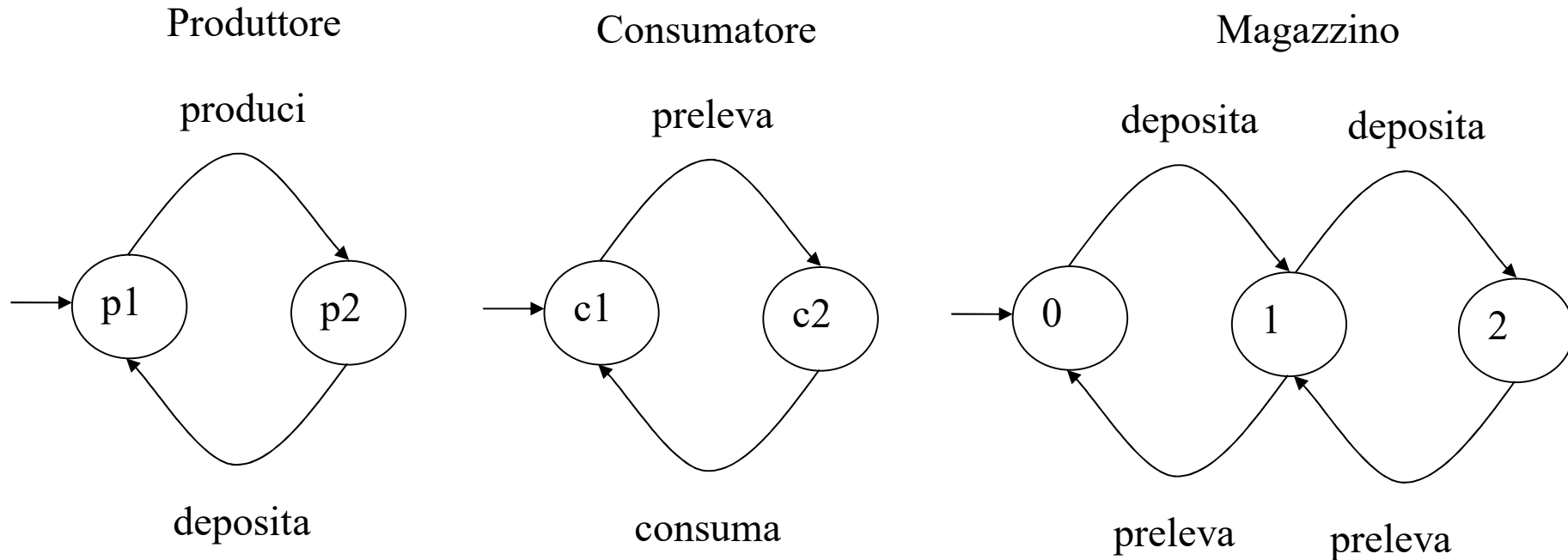


Legend:

$\xrightarrow{\text{<letter>}}$ is an abbreviation for a set of arrows labeled a, b, ..., z, A, ..., Z,

$\xrightarrow{\text{<digit>}}$ is an abbreviation for a set of arrows labeled 0, 1, ..., 9, respectively

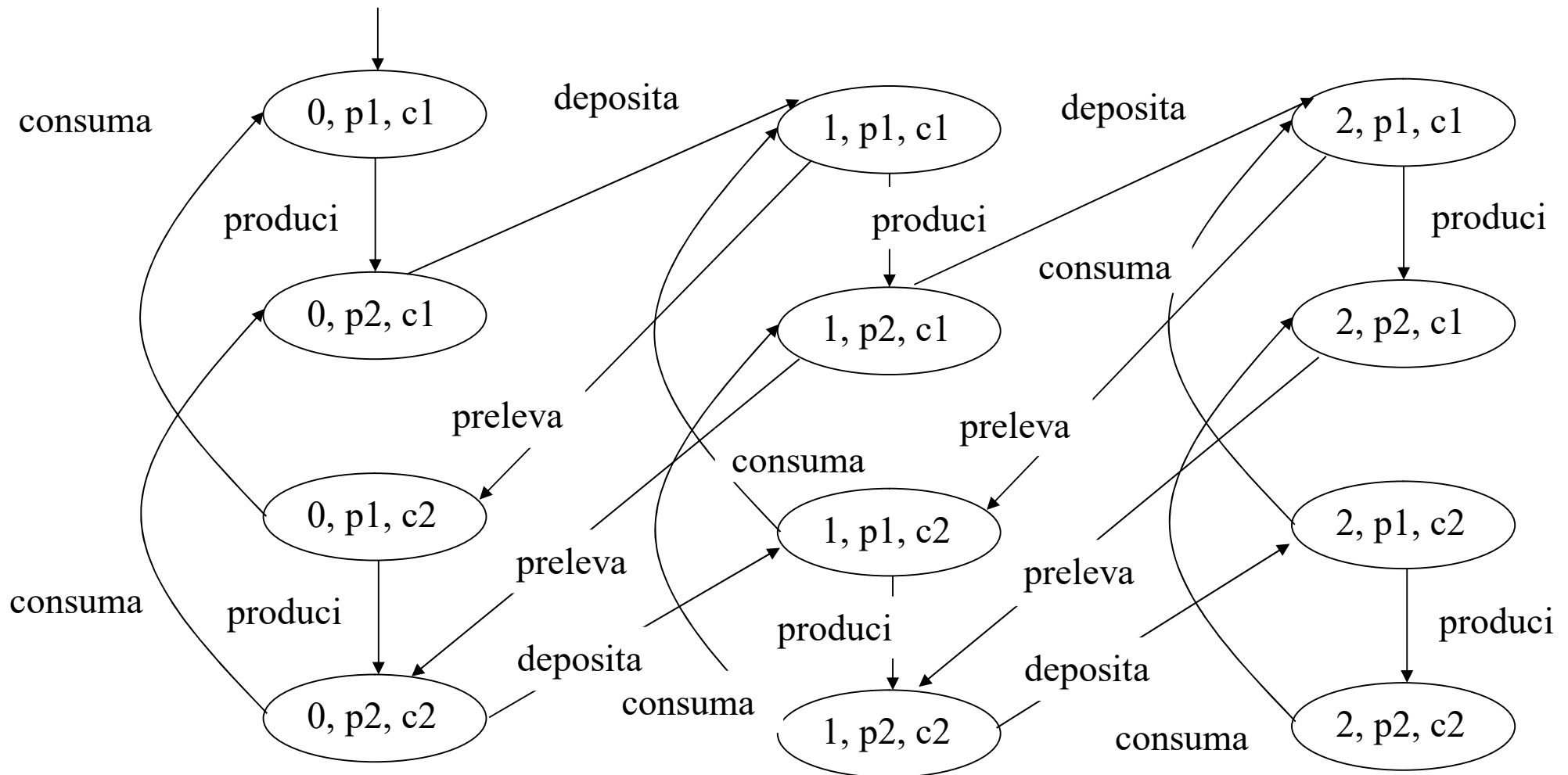
FSM: l'esempio dei processi produttore-consumatore



Limitazione

Esplosione degli stati quando si rappresentano sistemi concorrenti: date n FSM con k_1, k_2, \dots, k_n stati, la loro composizione è una FSM con $k_1 * k_2 * \dots * k_n$ stati (mentre vorremmo che fossero $k_1 + k_2 + \dots + k_n$)

FSM: l'esempio dei processi produttore-consumatore (cont.)

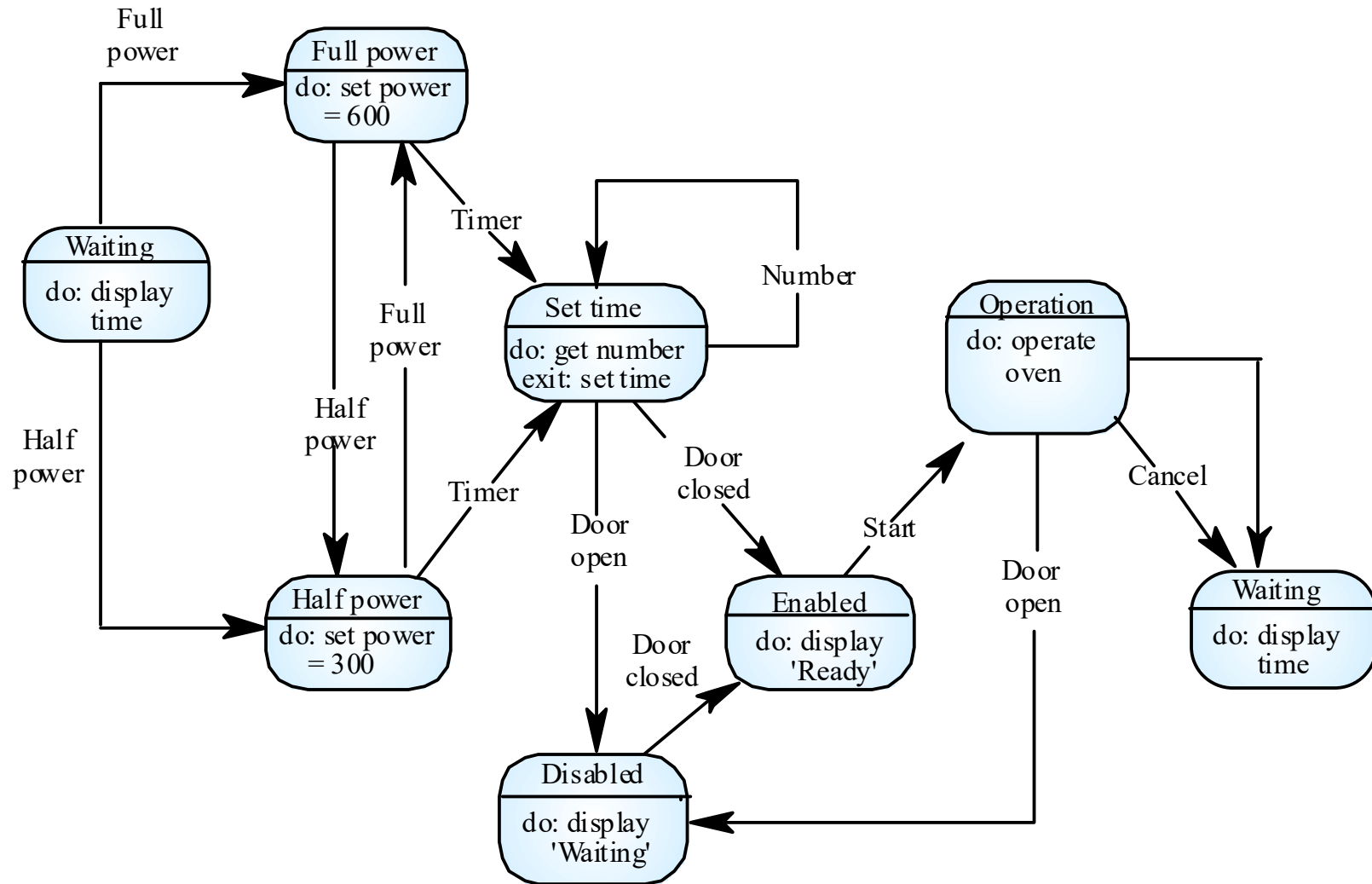


Macchine a stati finiti

Soluzione della limitazione delle FSM:

- Statechart (FSM cooperanti, usate in UML)
- Reti di Petri

Statechart



Reti di Petri (PN)

- I concetti di stato e transizione non sono più centralizzati ma distribuiti
- Descrizione naturale di sistemi asincroni
- L'evoluzione è non deterministica

Reti

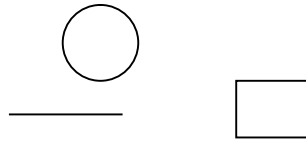
$$N = (P, T, F)$$

con

P: insieme finito di posti (places)

T: insieme finito di transizioni

F: relazione di flusso \longrightarrow



Vincoli:

$$(1) P \cap T = \emptyset$$

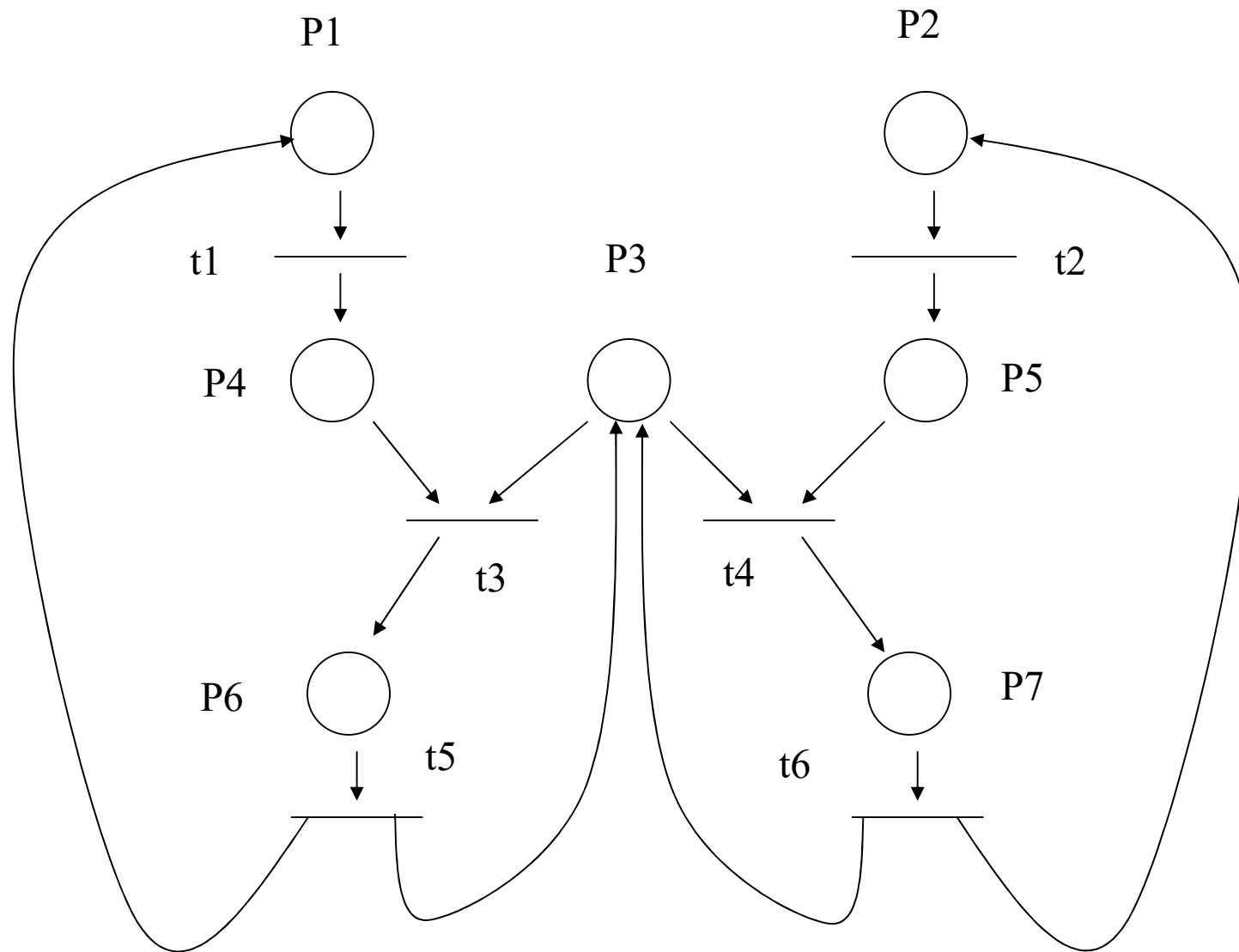
$$(2) P \cup T \neq \emptyset$$

$$(3) F \subseteq (P \times T) \cup (T \times P)$$

$$\text{Pre}(y \in P \cup T) = \{ x \in P \cup T \mid \langle x, y \rangle \in F \}$$

$$\text{Post}(x \in P \cup T) = \{ y \in P \cup T \mid \langle x, y \rangle \in F \}$$

Reti



PN

Modello di base: $PN = (P, T, F, W, M_0)$

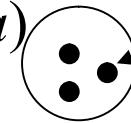
W: peso

M_0 : marcatura iniziale

Vincoli:

(4) $W: F \rightarrow N - \{0\}$, valore di default di W è 1 $\xrightarrow{4}$

(5) $M_0: P \rightarrow N$ (una funzione $M: P \rightarrow N$ è chiamata *marcatura*)



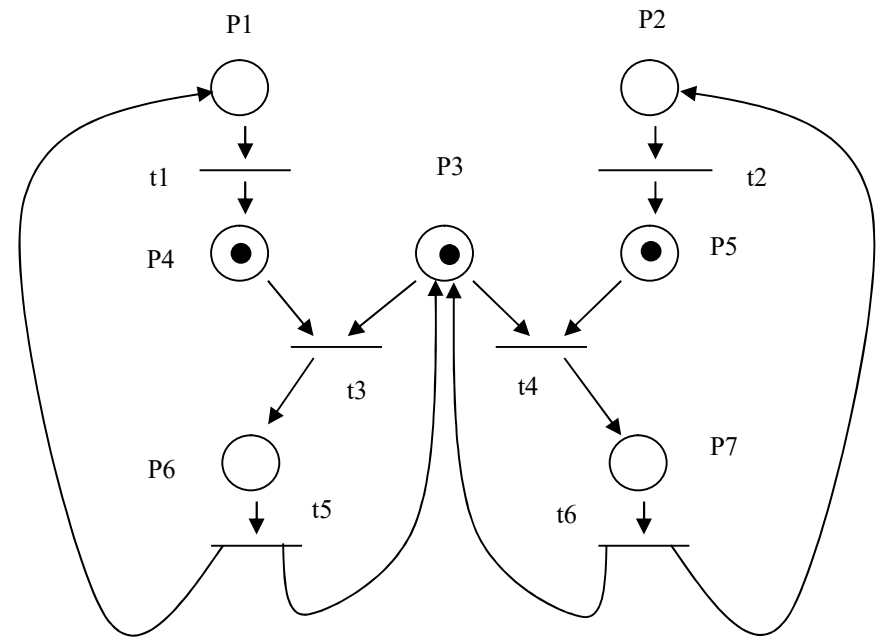
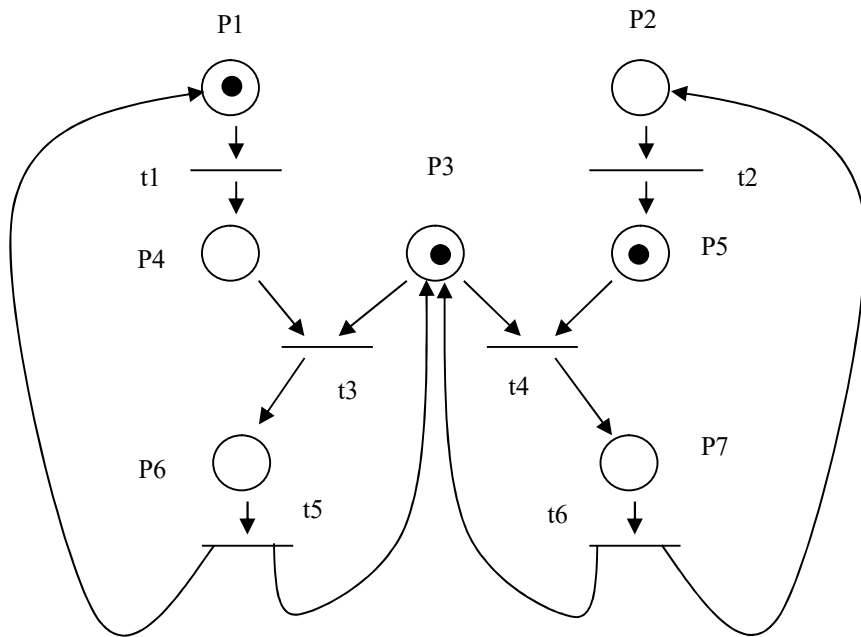
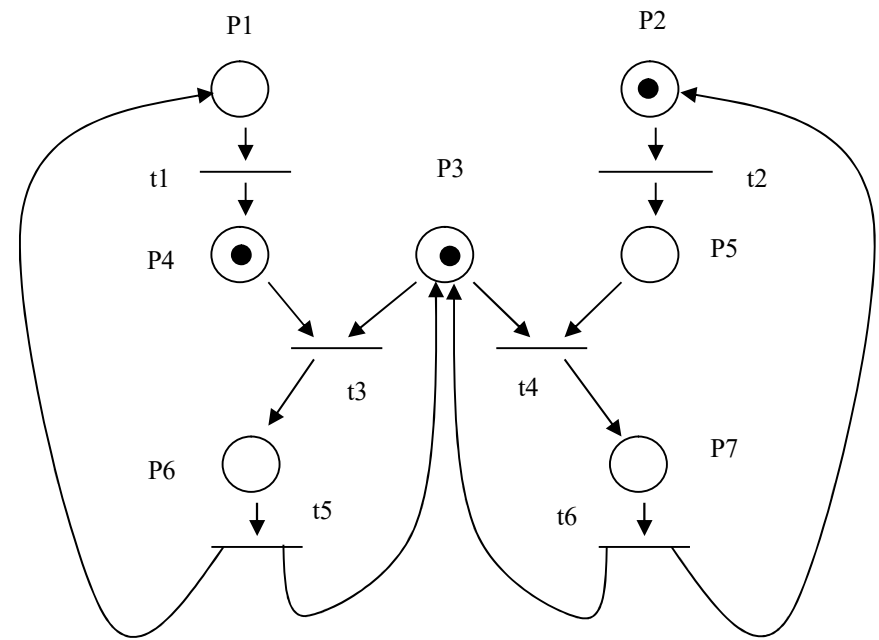
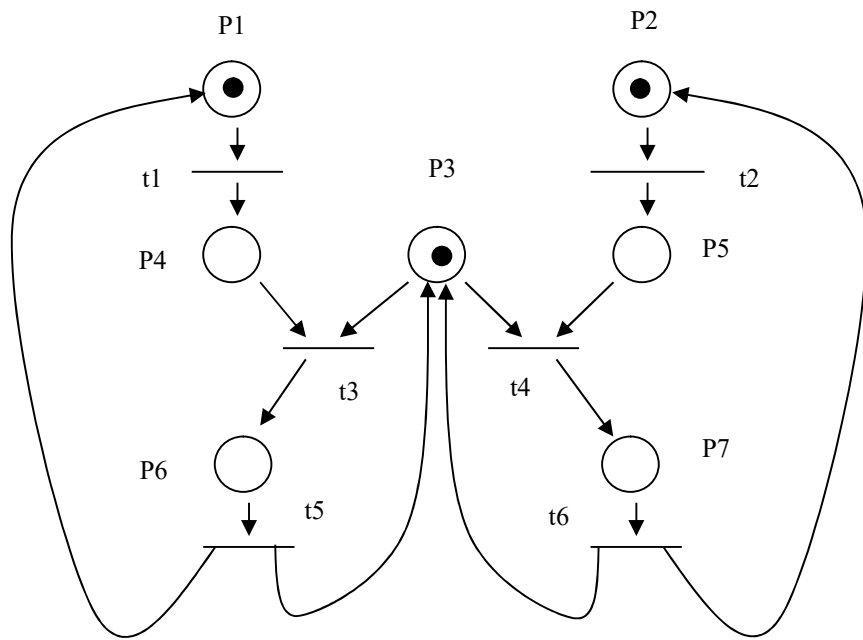
token

- Una marcatura è una rappresentazione di uno “stato” della rete
- La *semantica* definisce l’insieme delle possibili marcature prossime M' , data la marcatura corrente M
- *Evoluzione dinamica*: $M_0 \Rightarrow M_1 \Rightarrow M_2 \Rightarrow \dots$

PN: semantica

Descrizione informale (caso $W=1$)

- ◆ una transizione t è *abilitata* se c'è (almeno) un token in ogni $p \in \text{Pre}(t)$
- ◆ una transizione t abilitata può scattare, dando luogo a una nuova marcatura dove:
 - un token è cancellato da ogni $p \in \text{Pre}(t)$
 - un token è aggiunto a ogni $p \in \text{Post}(t)$
- ◆ se esistono più transizioni abilitate, la scelta della transizione da eseguire è nondeterministica



PN: semantica (cont.)

Descrizione formale (caso generale)

- ◆ Abilitazione di una transizione t data in corrispondenza della marcatura M (si scrive $M[t >]$):

$$\forall p \in \text{Pre}(t) \bullet M(p) \geq W(\langle p, t \rangle)$$

- ◆ Scatto di una transizione t ($M[t > M']$):

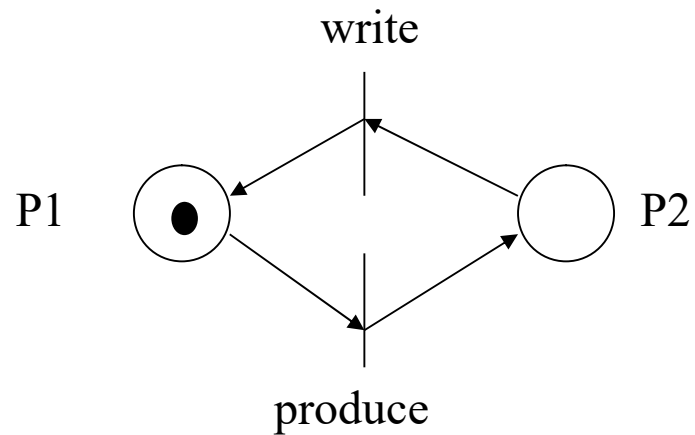
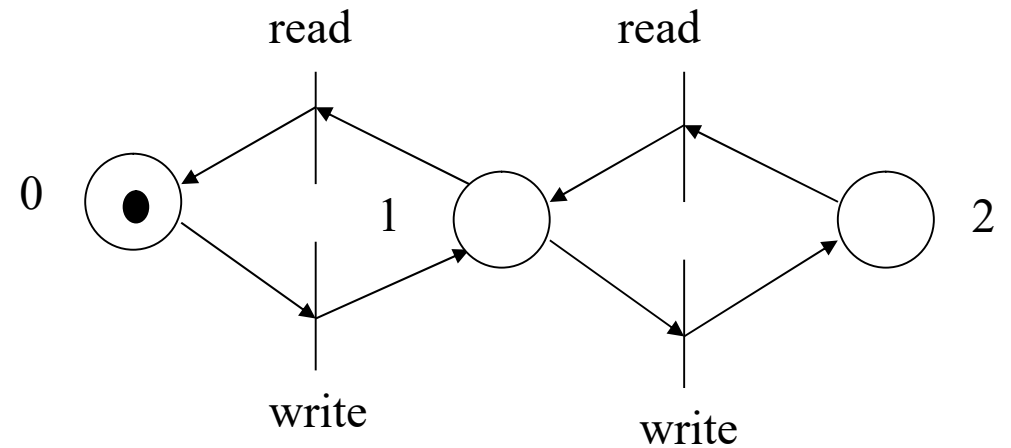
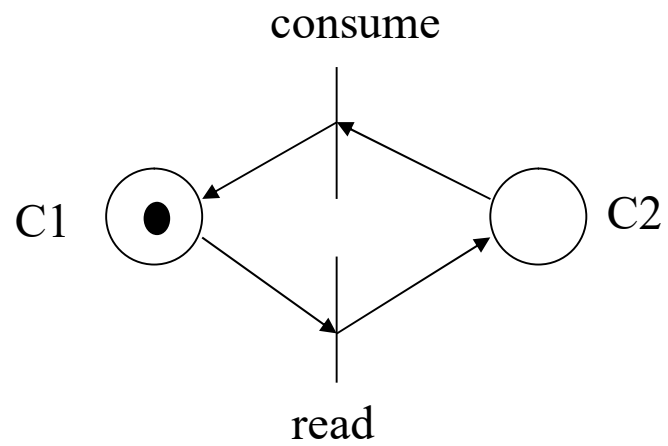
$$\forall p \in \text{Pre}(t) - \text{Post}(t) \bullet M'(p) = M(p) - W(\langle p, t \rangle)$$

$$\forall p \in \text{Post}(t) - \text{Pre}(t) \bullet M'(p) = M(p) + W(\langle t, p \rangle)$$

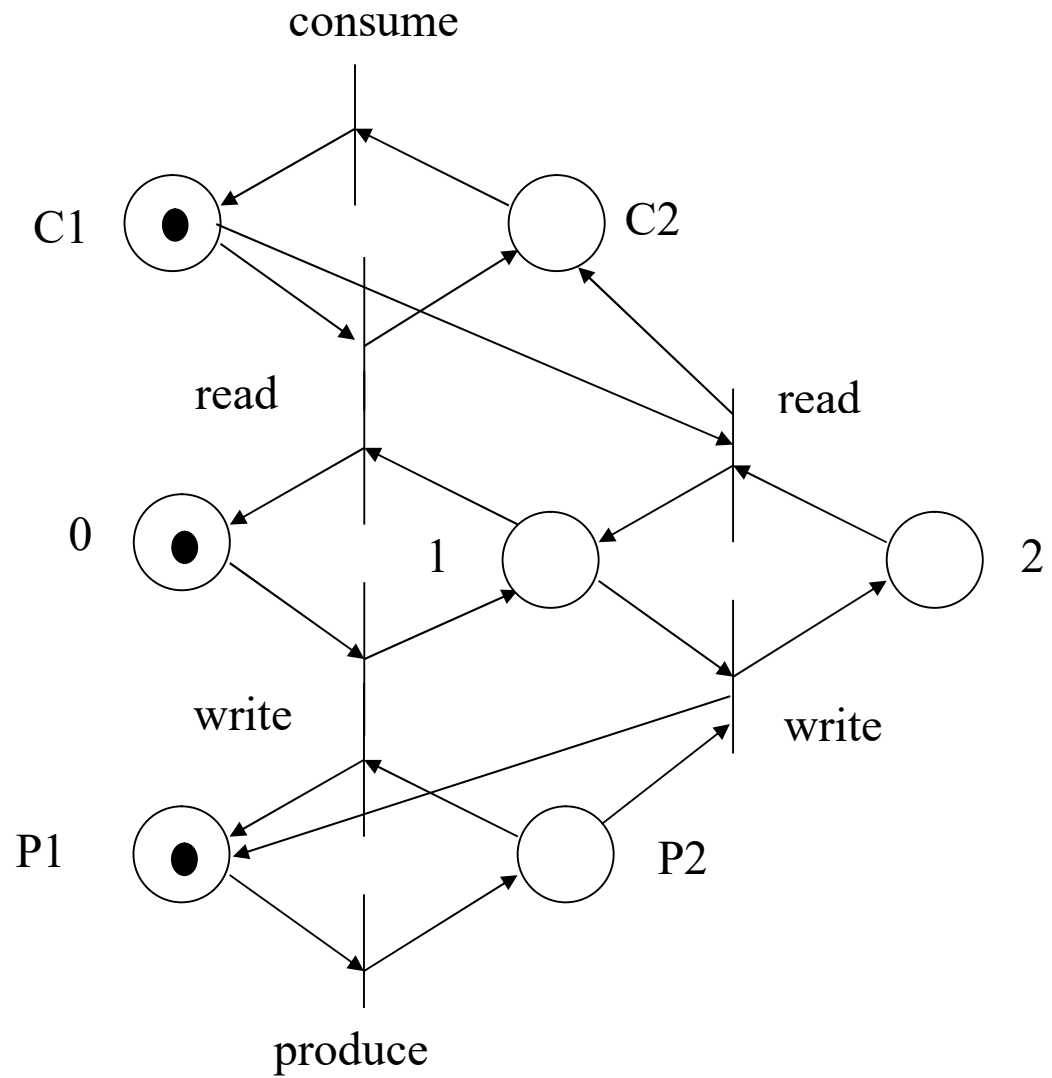
$$\forall p \in \text{Pre}(t) \cap \text{Post}(t) \bullet M'(p) = M(p) - W(\langle p, t \rangle) + W(\langle t, p \rangle)$$

$$\forall p \in P - (\text{Pre}(t) \cup \text{Post}(t)) \bullet M'(p) = M(p)$$

PN: l'esempio dei processi produttore-consumatore

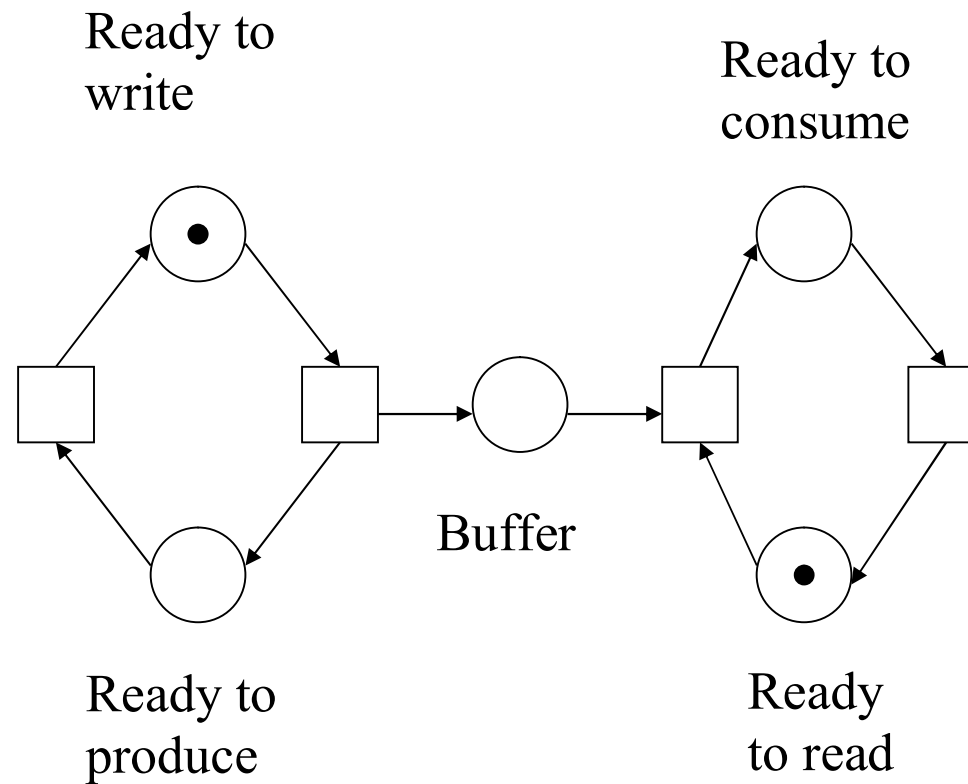


PN: l'esempio dei processi produttore-consumatore (cont.)

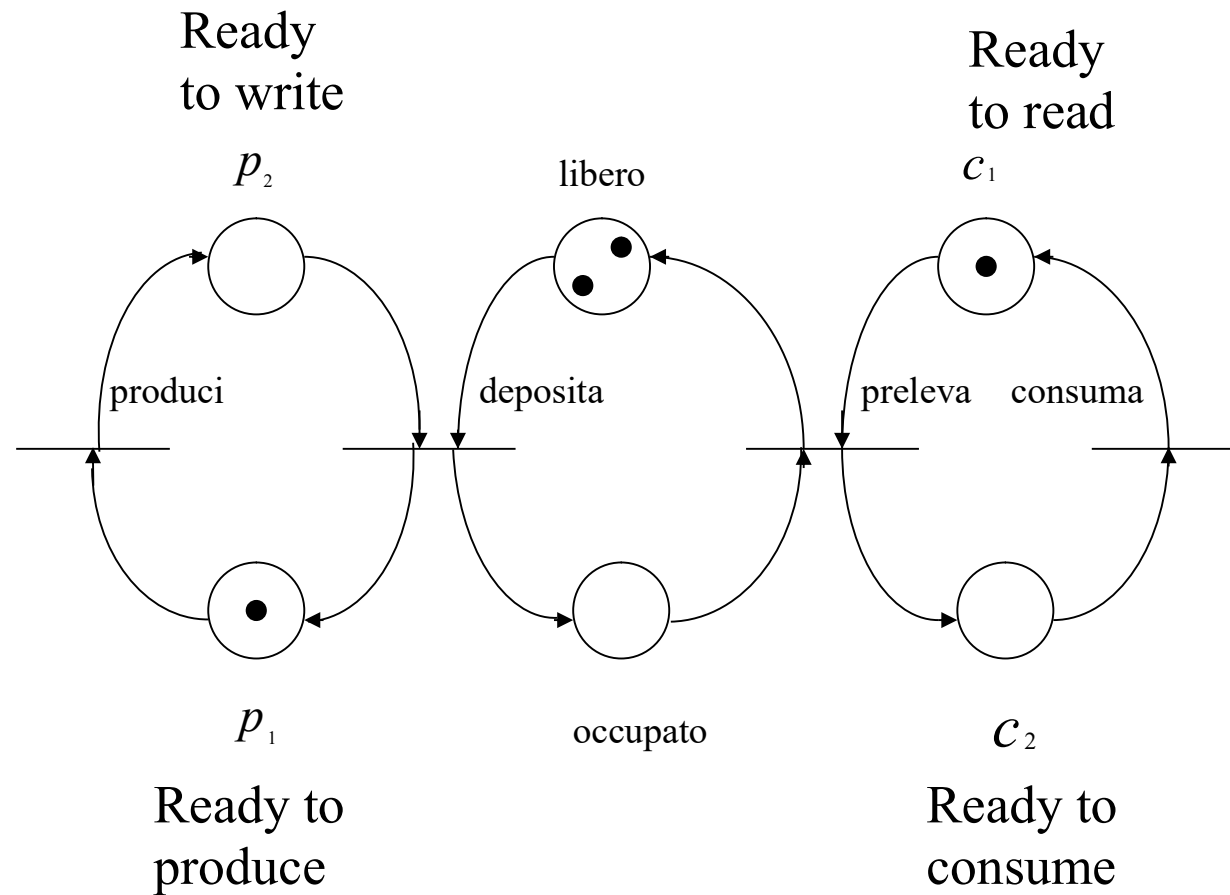


PN: processi produttore-consumatore con buffer illimitato

Una rete nei cui posti
si possono verificare
accumuli illimitati
di token si dice
illimitata



PN: processi produttore-consumatore con buffer finito

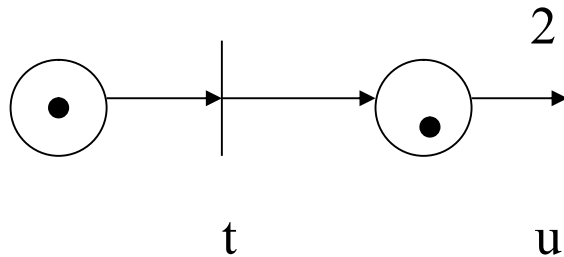


Sequenze e conflitti

Date due transizioni t e u , si definiscono

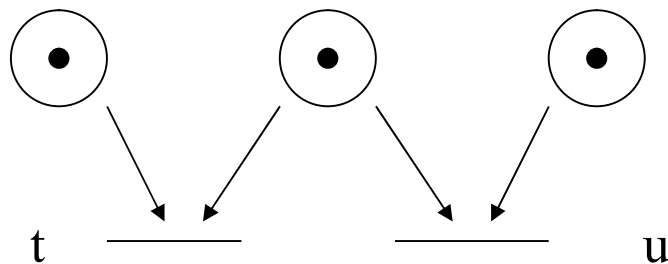
♦ Sequenza:

$$M[t] \wedge \neg M[u] \wedge M[tu]$$



♦ Conflitto:

$$M[t] \wedge M[u] \wedge \exists p \in \text{Pre}(t) \cap \text{Pre}(u) \bullet M(p) < W(\langle p, t \rangle) + W(\langle p, u \rangle)$$

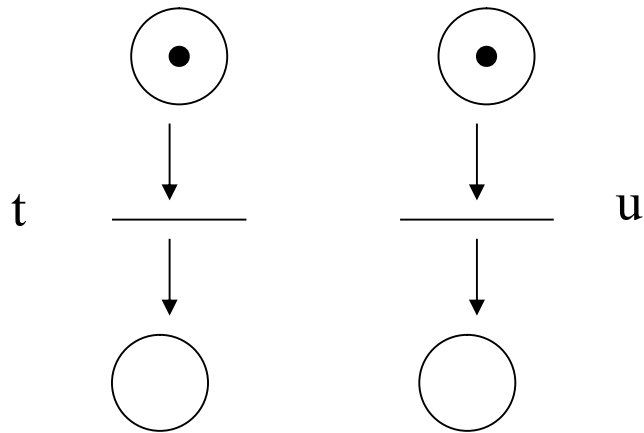


In presenza di conflitti, un processo può anche non accedere mai alle risorse necessarie alla sua evoluzione (*unfair scheduling*)

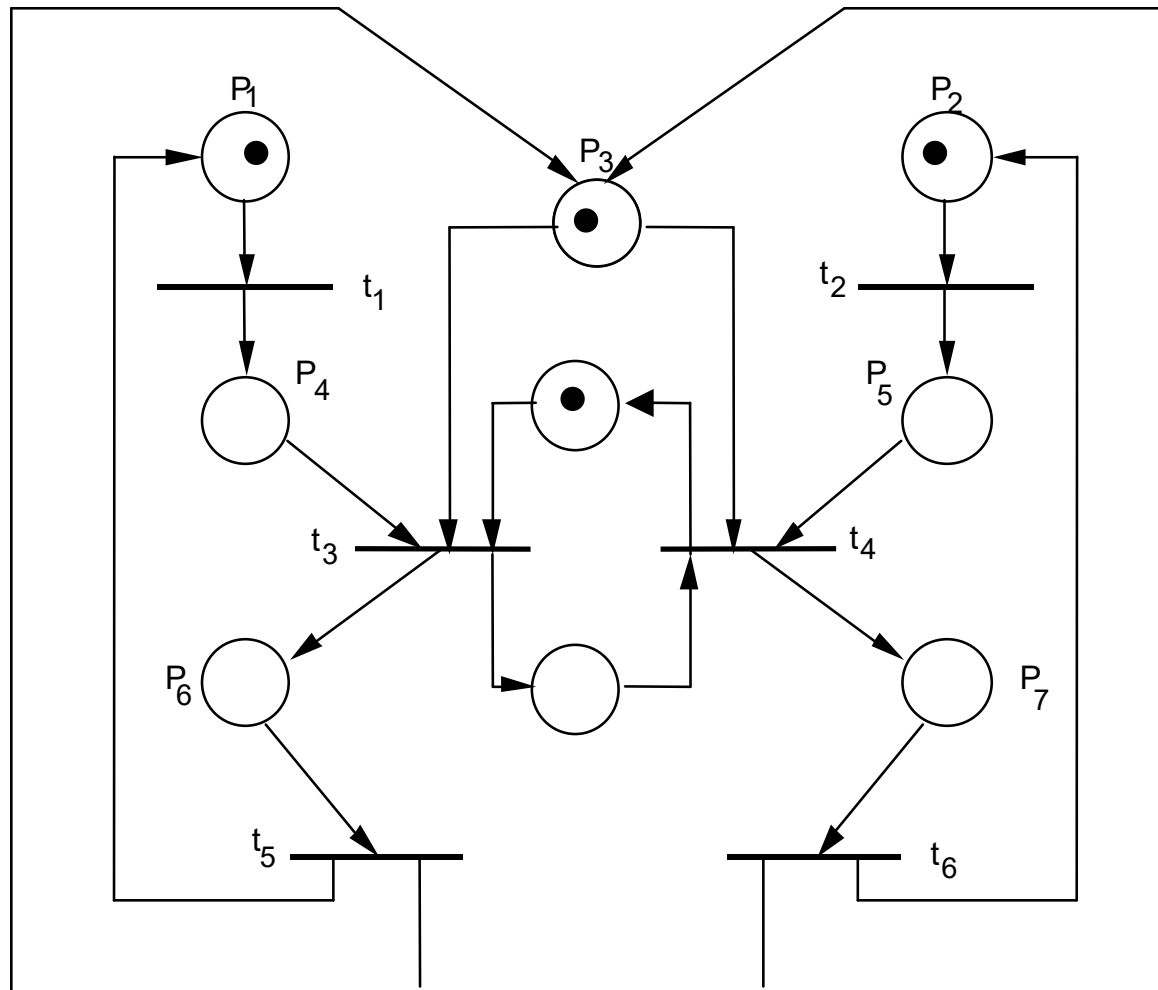
Concorrenza

♦ Concorrenza:

$$M[t] \wedge M[u] \wedge \forall p \in \text{Pre}(t) \cap \text{Pre}(u) \bullet M(p) \geq W(\langle p, t \rangle) + W(\langle p, u \rangle)$$



Fair scheduling

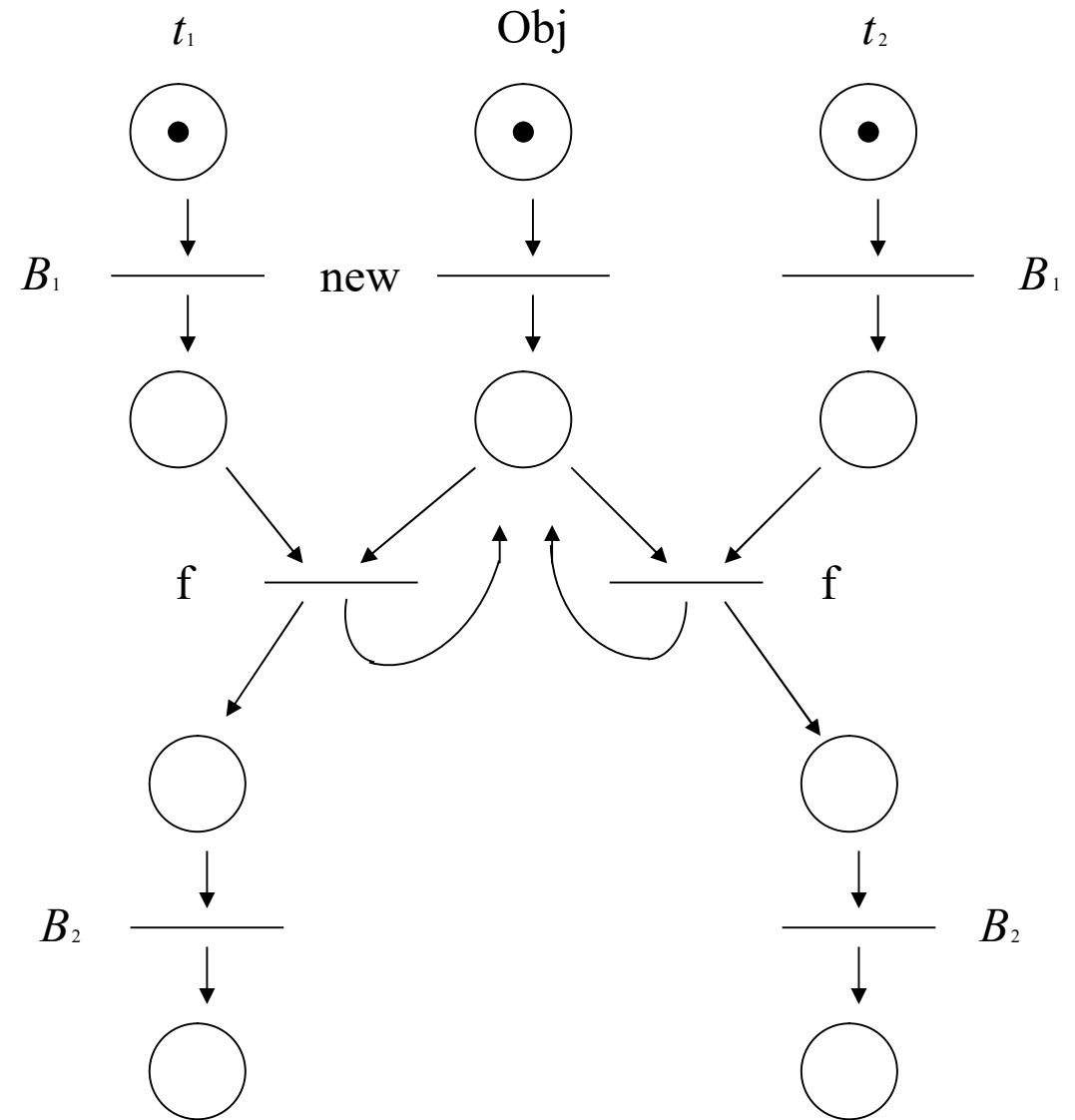


Produrre due nuove versioni della rete di Petri che garantiscano un *fair scheduling* in cui

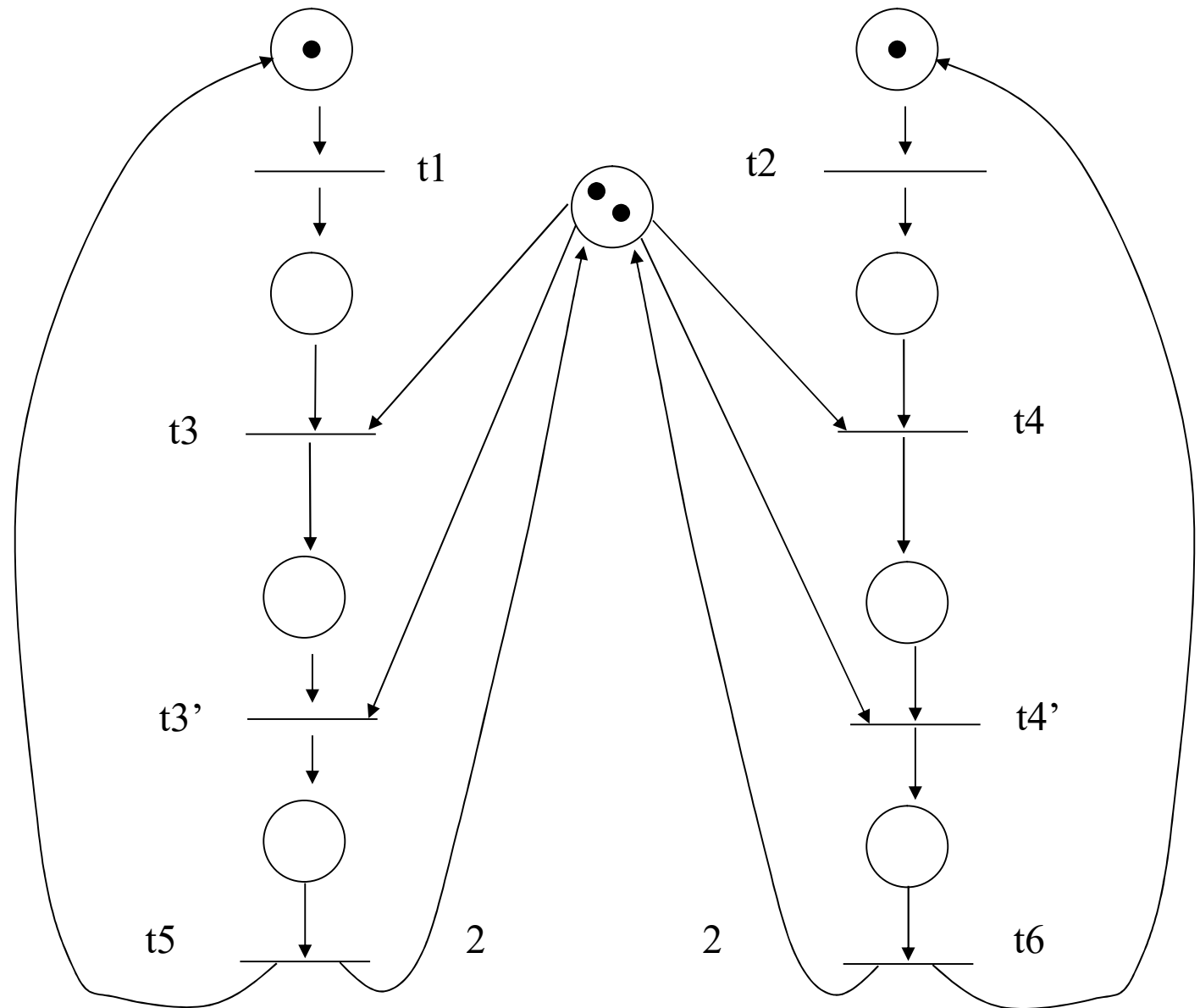
- ciclicamente prima il processo di sinistra si appropri della risorsa condivisa due volte e poi quello di destra una volta, oppure
- ciclicamente prima il processo di sinistra si appropri della risorsa condivisa tre volte e poi quello di destra due volte

Modellizzazione di metodi sincronizzati

◆ ad es. in Java

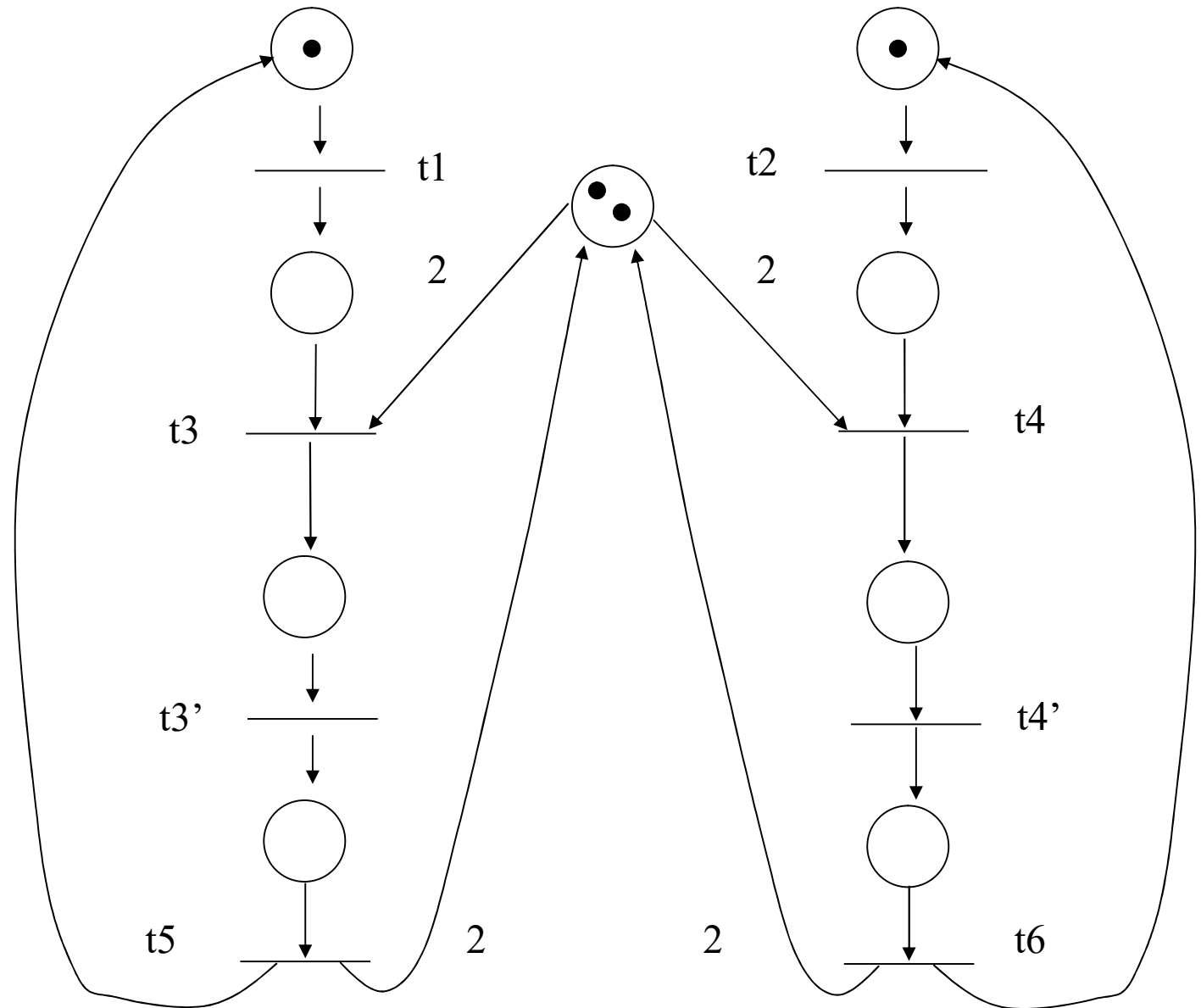


Deadlock



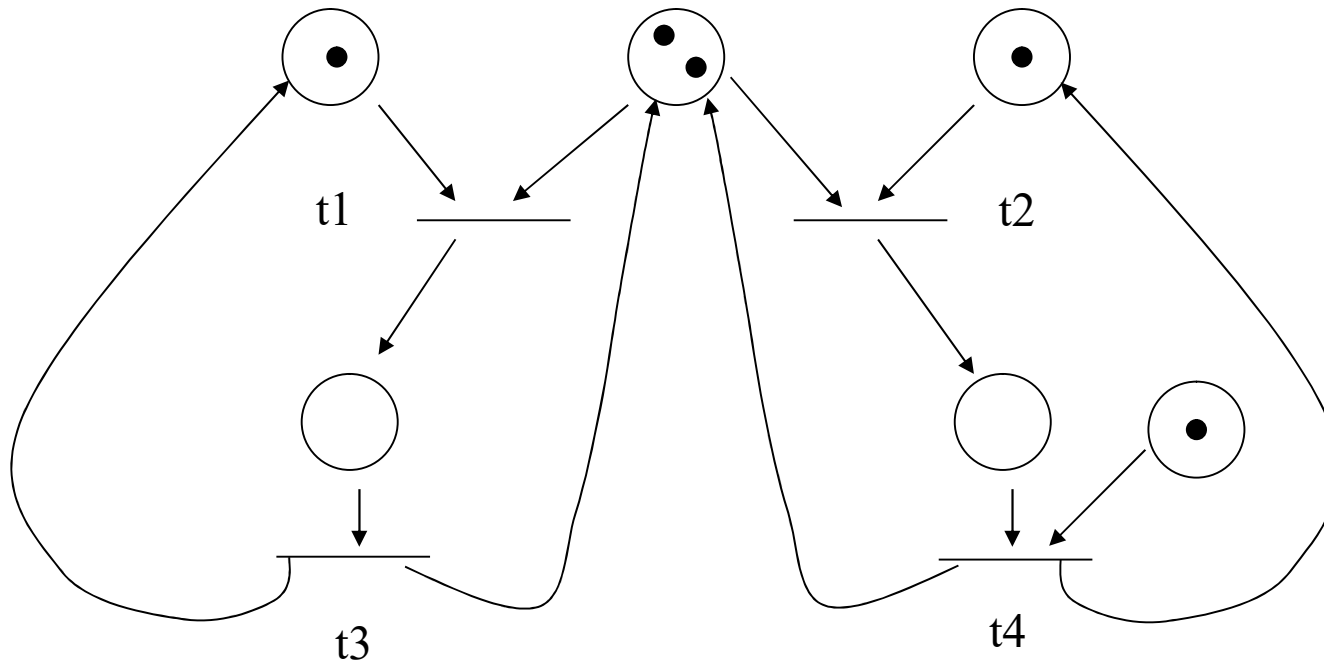
Nessun deadlock

Una rete priva
di deadlock si
dice *viva*



Starvation

Rete viva



Analisi di raggiungibilità

È tesa alla verifica delle proprietà

Consiste nel trovare le marcature raggiungibili a partire da una marcatura iniziale: problema decidibile ma di complessità esponenziale

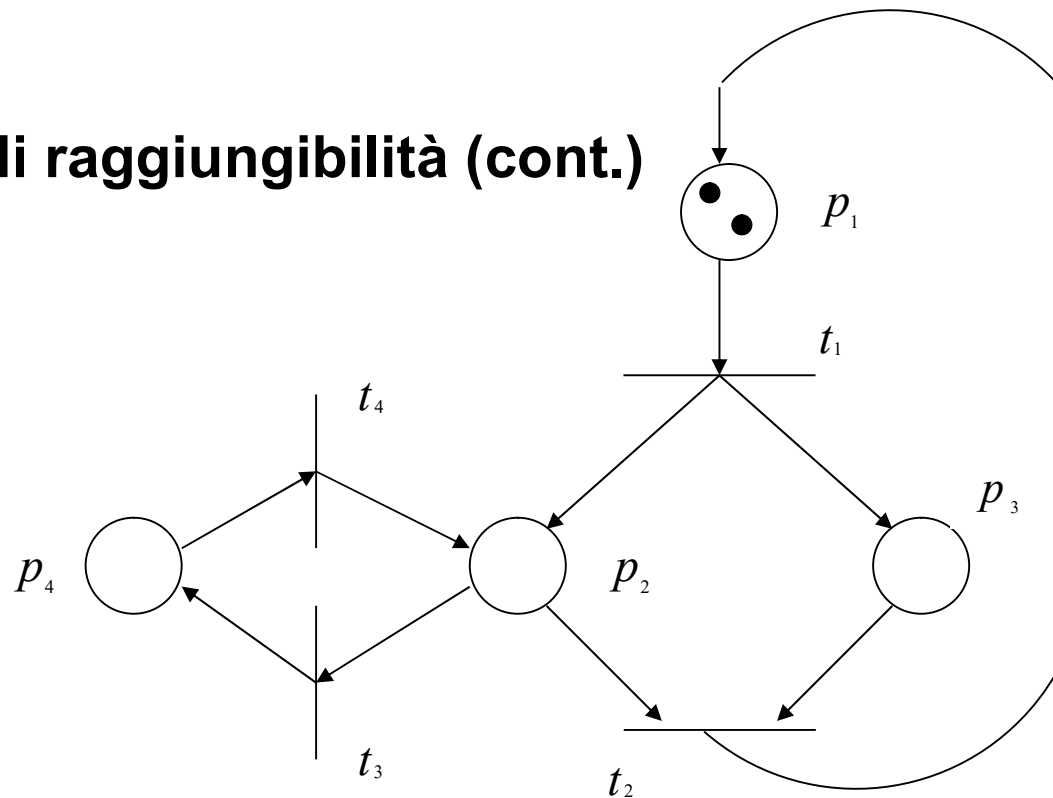
Raggiungibilità:

M' è raggiungibile in un passo se $\exists t \bullet M[t > M'$.

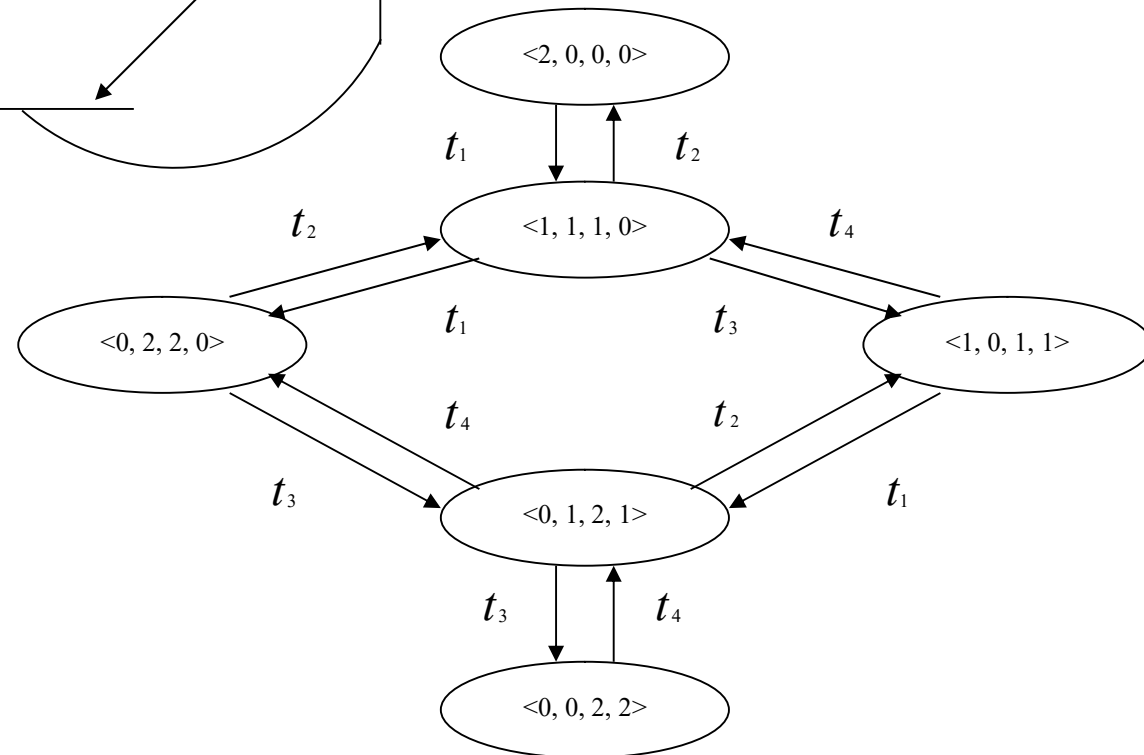
M' è raggiungibile se $\exists t_1 \dots t_k \bullet M[t_1 \dots t_k > M'$.

Rete illimitata = rete dotata di infinite marcature raggiungibili

Analisi di raggiungibilità (cont.)

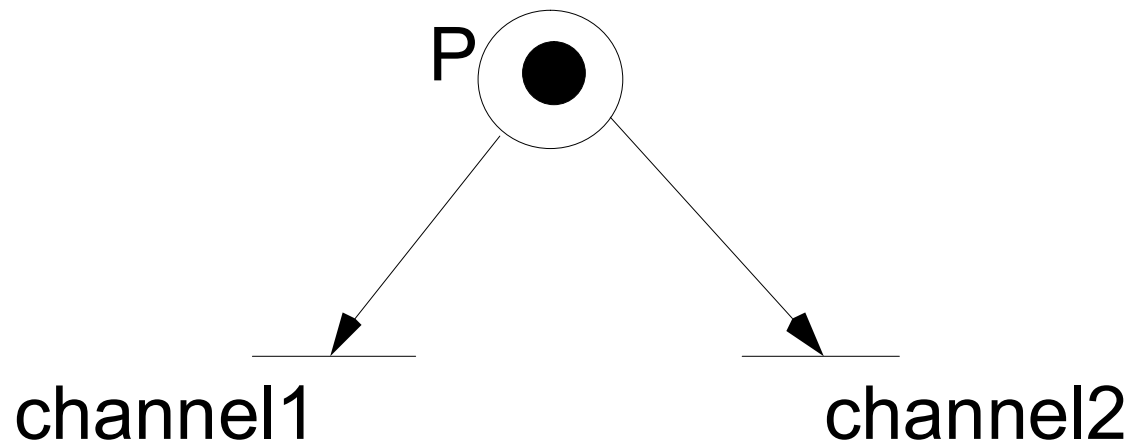


Grafo di
raggiungibilità:
nessun deadlock



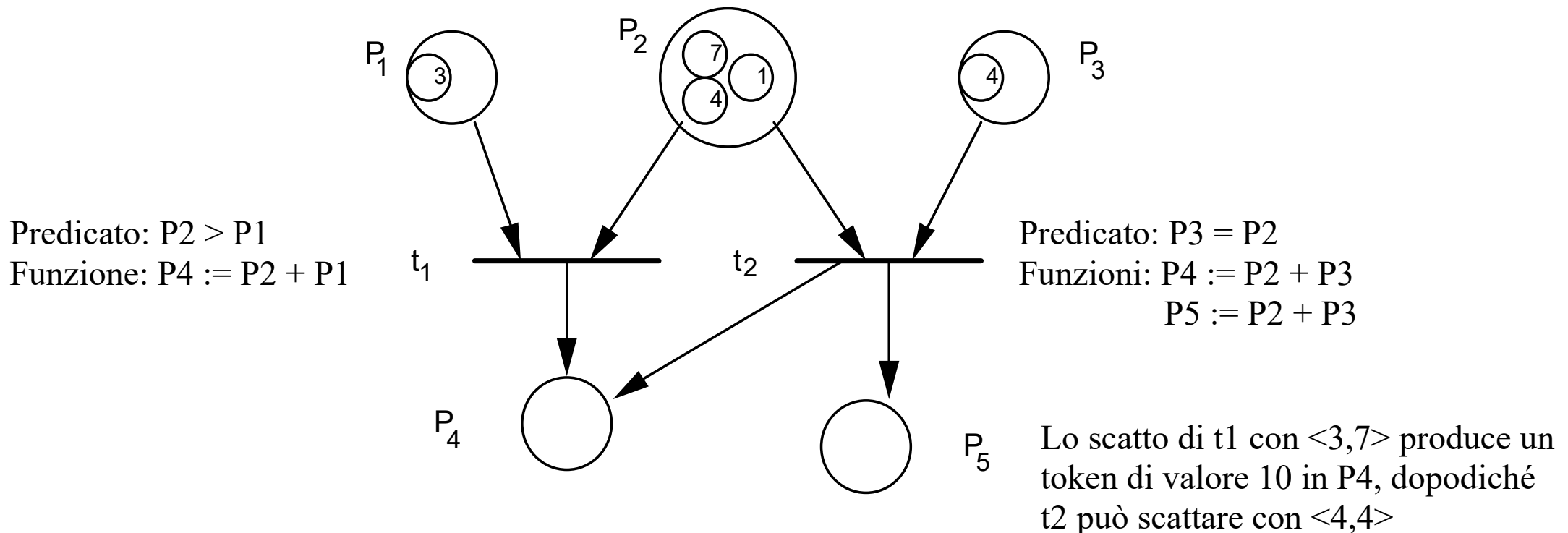
PN: limitazioni

Specifica non rappresentabile: un messaggio, a seconda del suo contenuto, viene inoltrato sul canale 1 oppure sul canale 2



Estensione delle PN dotando di valore i token

- A ogni transizione si associano un predicato e tante funzioni quanti sono gli elementi della relazione di flusso in uscita
- Il predicato rappresenta una condizione di abilitazione della transizione dipendente dai valori dei token nei posti predecessori della stessa
- Le funzioni definiscono i valori dei token da introdurre nei posti successori della transizione allo scatto della stessa



Estensione delle PN con l'aggiunta di priorità

- La priorità è una funzione $T \rightarrow N$
- Quando esistono più transizioni abilitate, lo scatto è consentito solo a quelle di priorità massima
- La scelta di quale transizione scatti fra quelle abilitate di priorità massima è nondeterministica

PN temporizzate

Reti di Merlin&Farber

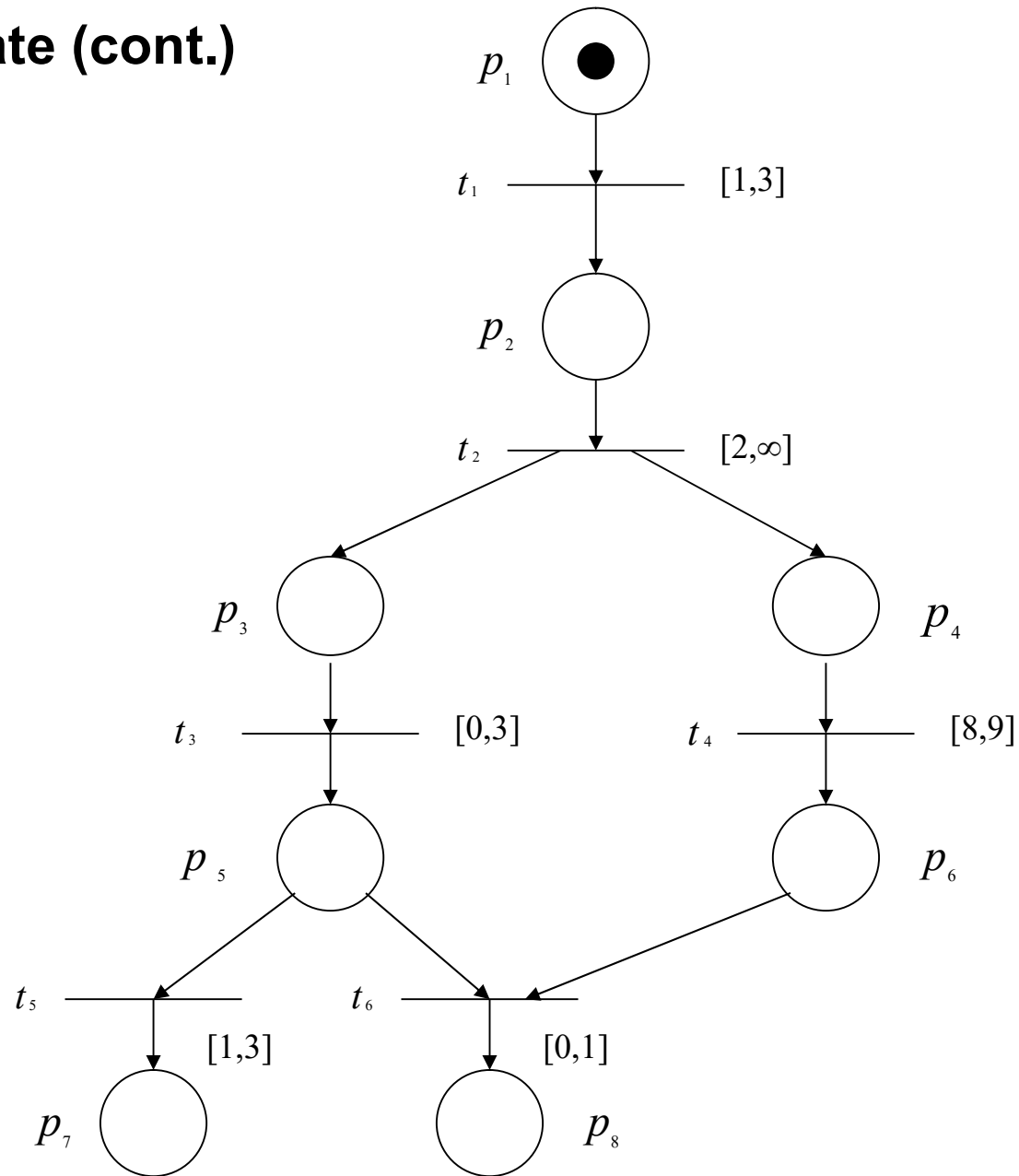
- a ogni transizione è associato un intervallo $[t_{\min}, t_{\max}]$ che rappresenta il tempo minimo/massimo per lo scatto \rightarrow cambia la condizione di abilitazione di una transizione



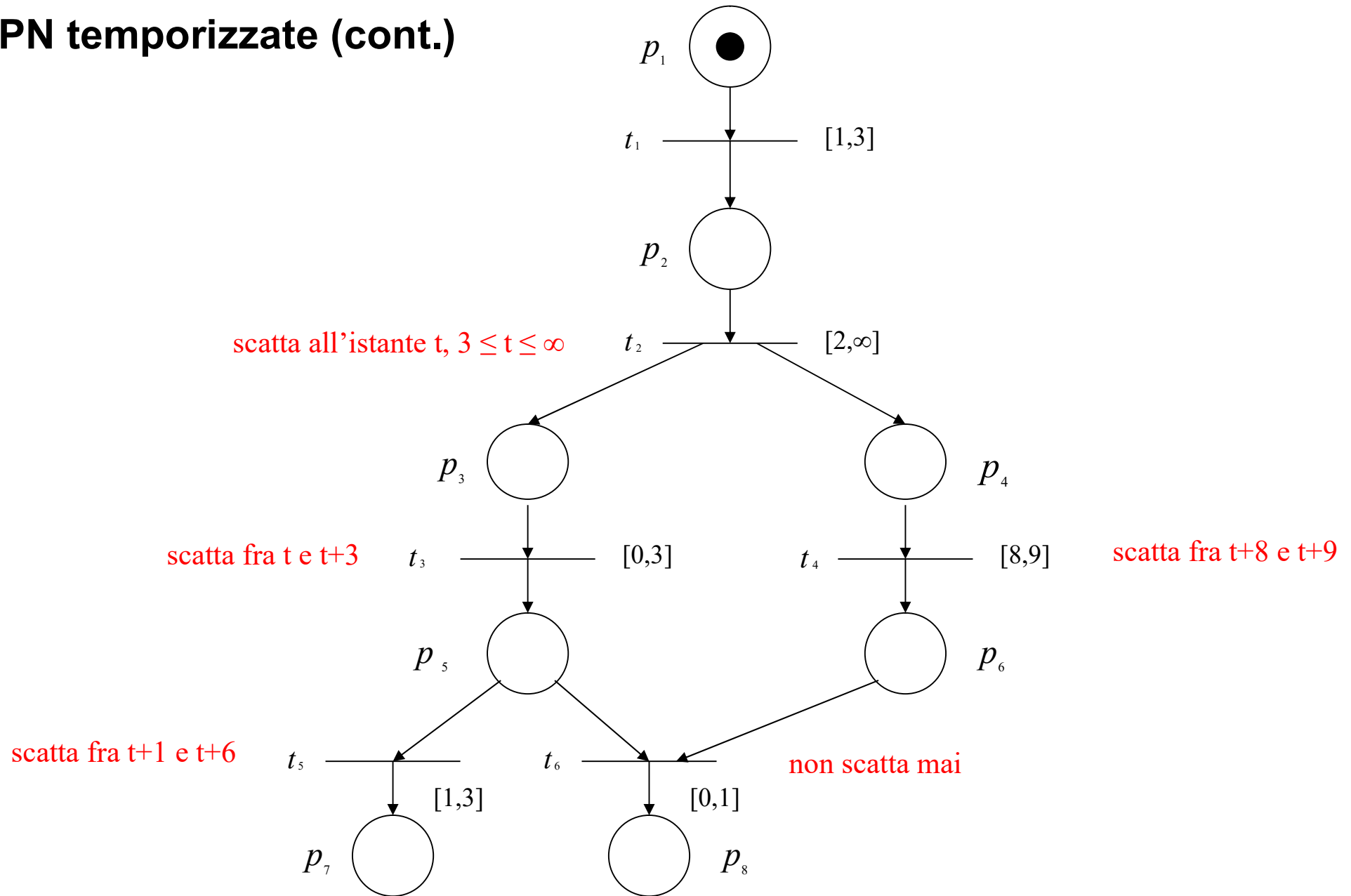
se una transizione è abilitata nel senso originale all'istante t , essa DEVE scattare fra l'istante $t + t_{\min}$ e $t + t_{\max}$, a meno che non venga disabilitata dallo scatto di un'altra transizione

- si presuppone l'esistenza di un orologio globale
- la rete è di tipo stocastico se per i tempi di scatto delle transizioni si forniscono le distribuzioni di probabilità

PN temporizzate (cont.)

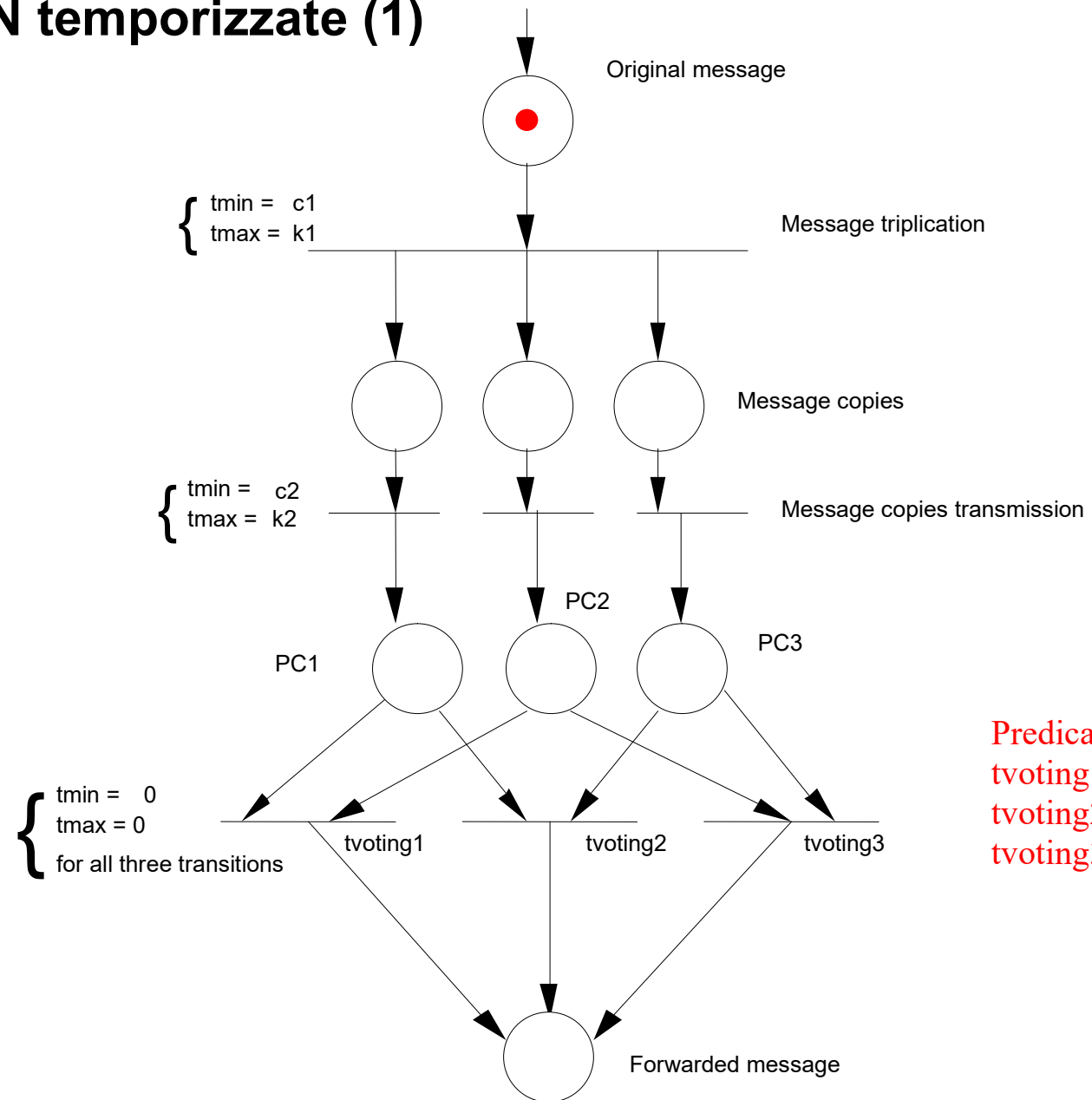


PN temporizzate (cont.)



Specifica mediante PN temporizzate (1)

messaggio inoltrato
non appena sono
state ricevute due
copie identiche

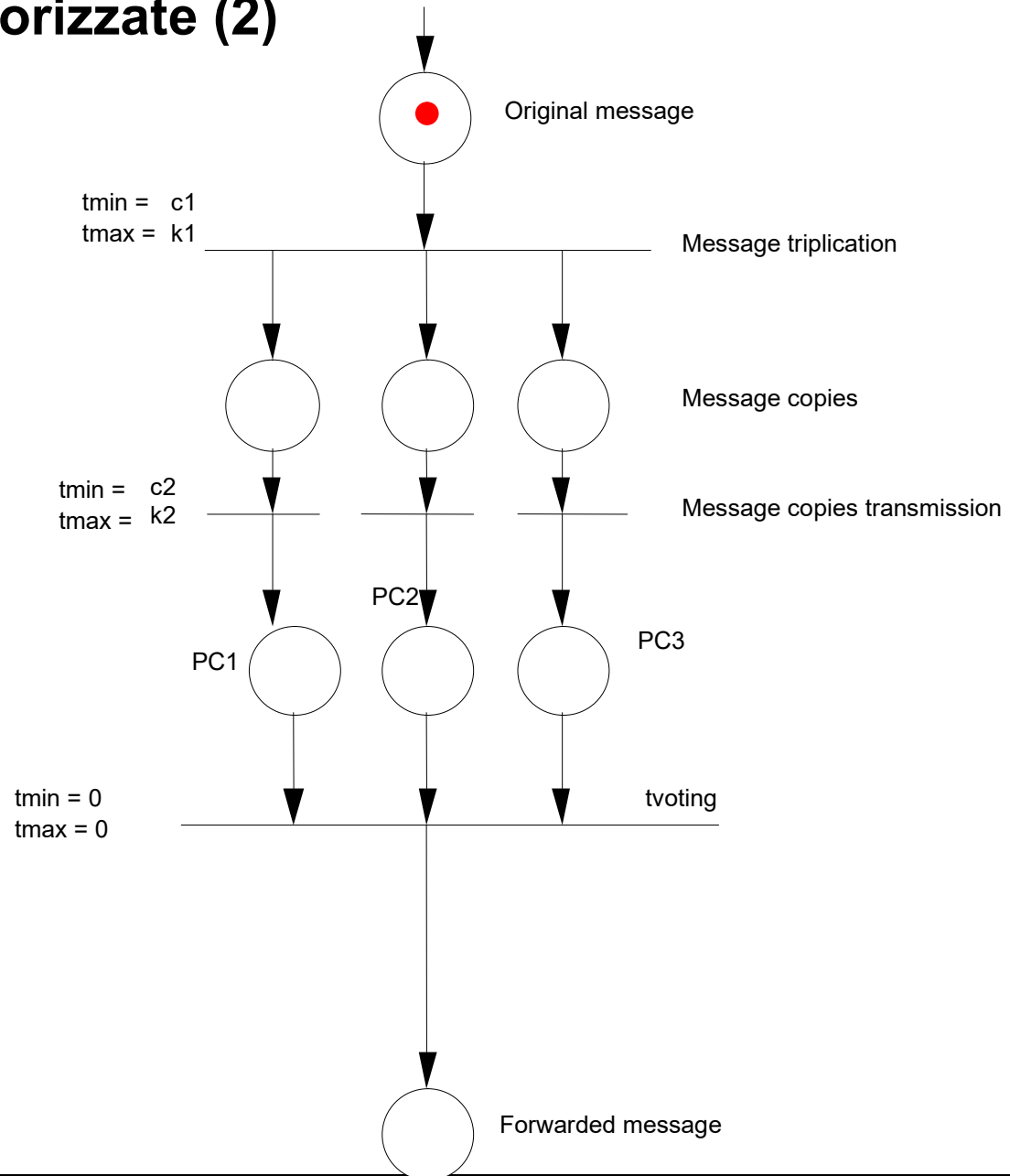


Predicati
tvoting1: PC1=PC2
tvoting2: PC1=PC3
tvoting3: PC2=PC3

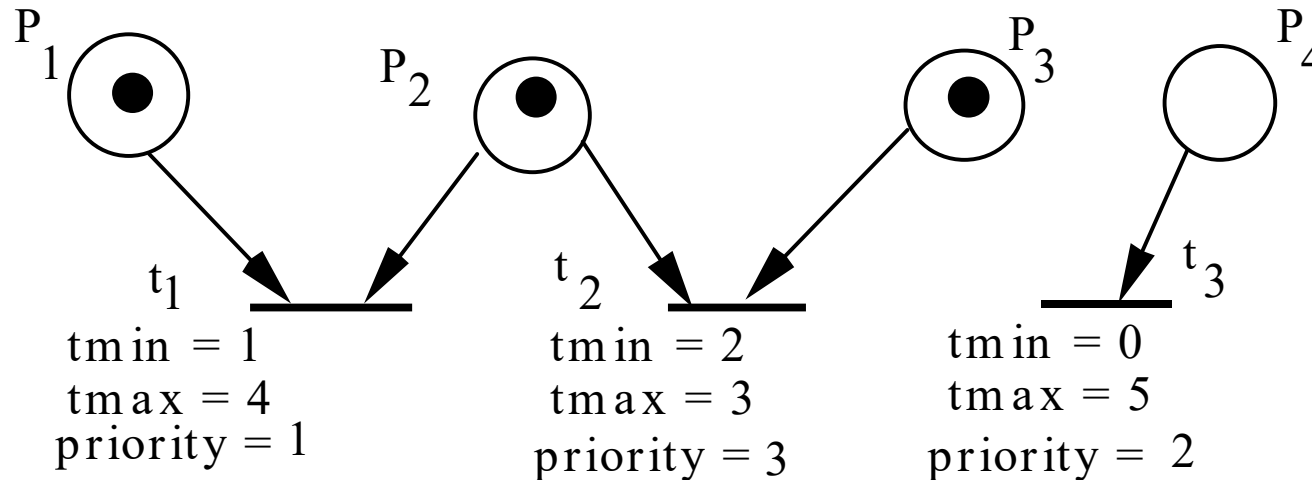
Specifica mediante PN temporizzate (2)

messaggio inoltrato
dopo che sono state
ricevute tutte e tre le
copie e solo se
almeno due sono
identiche

Predicato
tvoting: $PC1=PC2$ or $PC1=PC3$ or $PC2=PC3$



Combinazione di due estensioni (priorità e temporizzazione)





Assunto che i tre token in figura siano stati depositati nei rispettivi posti all'istante 0, la transizione t_1 , se non scatta prima dell'istante 2, non può più scattare perché la transizione t_2 ha priorità più alta

Se, data la marcatura in figura, all'istante 1 appare un token in P_4 , prima dell'istante 2 possono (ma non devono necessariamente) scattare sia t_3 sia t_1 , però solo in quest'ordine relativo (t_1 non può scattare prima di t_3 perché ha priorità più bassa)

Diagrammi di attività

- Combinano idee tratte da molte tecniche diverse (diagrammi degli eventi, modellazione di stato SDL, modellazione di workflow, reti di Petri)
- Costituiscono un argomento complesso (e lo sono diventati ancor di più in UML 2) → sono una delle parti peggio comprese di UML
- Sono focalizzati sulla rappresentazione della logica comportamentale
- Sono simili ai diagrammi di flusso (flowchart) ma, a differenza di questi, supportano (e incoraggiano) anche la rappresentazione di elaborazioni parallele, il che è il loro punto di forza
- Il loro punto di debolezza è l'incapacità di rappresentare l'accesso condiviso ai dati, in cui risiede gran parte della complessità della programmazione concorrente
- Mostrano la sequenza generale di attività svolte da più oggetti in casi d'uso diversi

Diagrammi di attività (cont.)

Descrizione dell'attività mediante costrutti di rappresentazione di comportamenti condizionali (decisione e merge ) e paralleli (fork e join )

Attività = composizione di azioni; processo nel mondo reale (es. processi di business e workflow) o esecuzione di una procedura sw (es. esecuzione di un metodo)

Diagramma di attività



Elementi	Sintassi	Semantica
Nodo iniziale 	Rispettivamente come il punto di partenza e quello di arrivo dei diagrammi di stato	Nell'ambito dell'Ing. del sw, il nodo iniziale corrisponde all'invocazione di un programma o di una procedura
Nodo finale 	I flussi entranti nel nodo finale sono in merge implicito	Il fatto che “i flussi entranti nel nodo finale” siano “in merge implicito” significa che l'esecuzione termina non appena un flusso raggiunga il nodo finale

Diagramma di attività (cont.)


Elementi	Sintassi	Semantica
<p>Azione</p> 	<p>Rettangolo con angoli arrotondati</p> <ul style="list-style-type: none"> • contenente il <i>nome</i> dell'azione, eventualmente accompagnato da (<i>classe::metodo</i>) oppure da un'icona rastrello  • avente uno o più flussi in ingresso e un flusso in uscita 	<p>Il rastrello indica che l'azione è stata scomposta mediante un sottodiagramma di attività (non visualizzato)</p> <p>L'azione viene eseguita solo quando tutti i flussi in ingresso l'hanno raggiunta (join implicito)</p> <p>Per evitare fraintendimenti, si suggerisce di disegnare sempre un singolo flusso di ingresso per ogni azione e di esplicitare sempre tutti i join e tutti i merge</p>

Diagramma di attività (cont.)


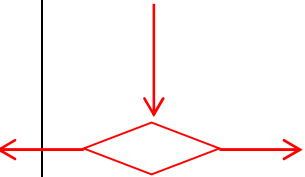
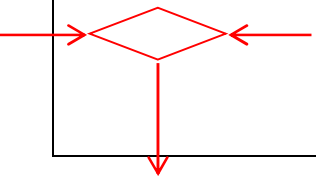
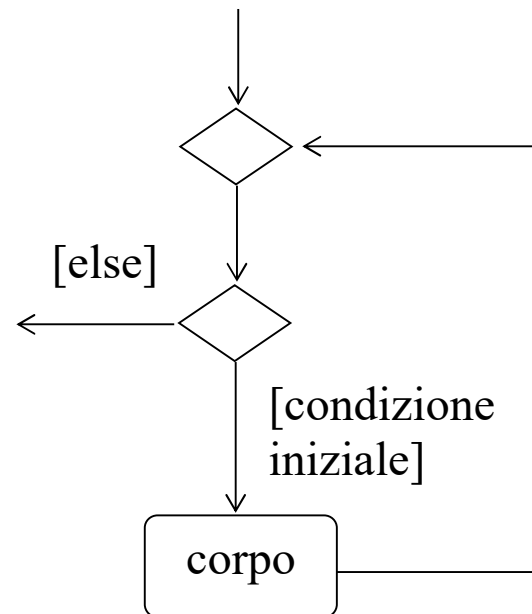
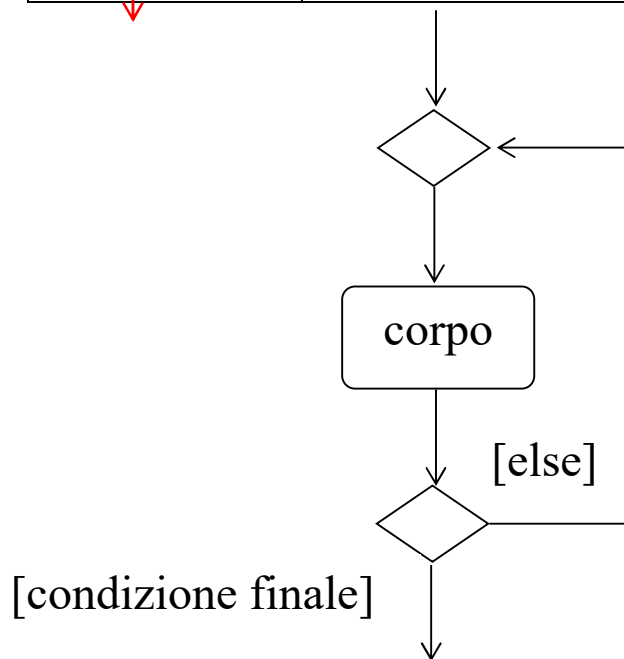
Elementi	Sintassi	Semantica
Flusso (o arco)	Freccia a linea continua e punta biforcuta, opzionalmente dotata di etichetta (nome dell'arco) 	Se uscente da un'azione, scatta al completamento della stessa senza bisogno di alcun evento trigger
Decisione (<i>branch</i> in UML 1) 	<ul style="list-style-type: none"> • Rombo con un singolo flusso entrante e due o più flussi uscenti, dove ciascun arco uscente è dotato di una <i>[condizione]</i> mutuamente esclusiva rispetto alle altre • Globalmente le condizioni devono essere esaustive • La condizione <i>[else]</i> rappresenta il caso in cui tutte le altre condizioni non sono verificate 	Quando scatta l'arco entrante, si segue il solo flusso uscente la cui condizione è vera

Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Merge (o giunzione) 	<ul style="list-style-type: none"> Rombo con due o più flussi entranti e un flusso uscente Termina un blocco condizionale cominciato con una decisione 	Il flusso d'uscita entra in esecuzione non appena uno dei flussi d'ingresso ha raggiunto il merge Le iterazioni sono ottenibili combinando azioni, decisioni e merge



Esercizio:
disegnare uno
stralcio di
diagramma UML
di attività per
rappresentare un
ciclo a conteggio

Diagramma di attività (cont.)

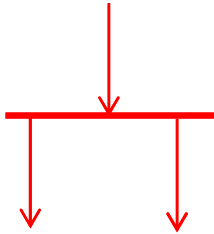
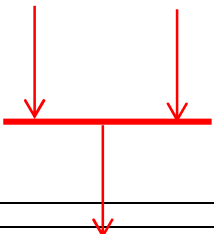
Elementi	Sintassi	Semantica
Fork (o divisione) 	<ul style="list-style-type: none"> Barra (non necessariamente) orizzontale spessa con un flusso entrante e due o più flussi uscenti Se un flusso uscente da un fork entra direttamente in un altro fork, quest'ultimo si può omettere, facendo uscire i suoi archi uscenti direttamente dal primo fork 	<ul style="list-style-type: none"> Quando scatta l'arco entrante, si seguono in parallelo (in modo concorrente) tutti i flussi uscenti L'ordine relativo secondo il quale vengono svolte le azioni dei flussi uscenti è irrilevante: il diagramma specifica le regole essenziali di ordinamento, quelle che non possono essere violate
Join (o unione) 	Barra (non necessariamente) orizzontale spessa con due o più flussi entranti e un solo flusso uscente	Punto di sincronizzazione: il flusso uscente può essere seguito solo quando tutti i flussi in entrata hanno terminato l'esecuzione

Diagramma di attività (cont.)

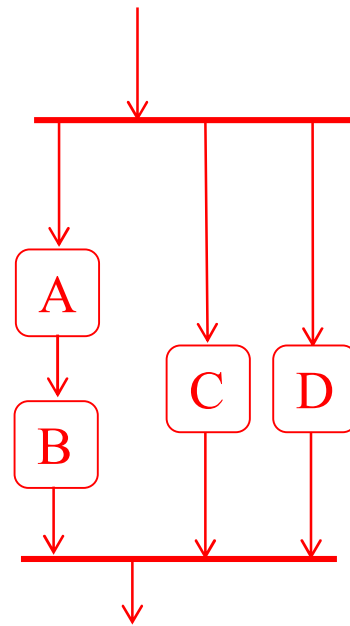
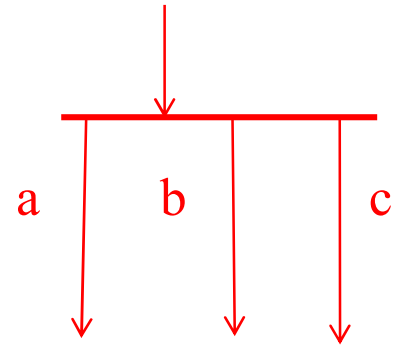
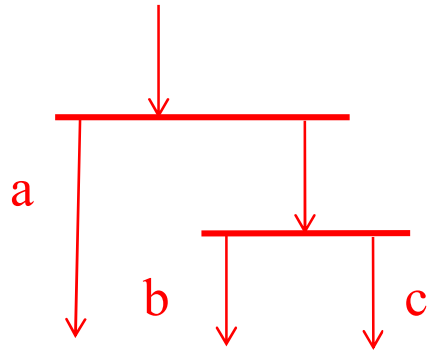
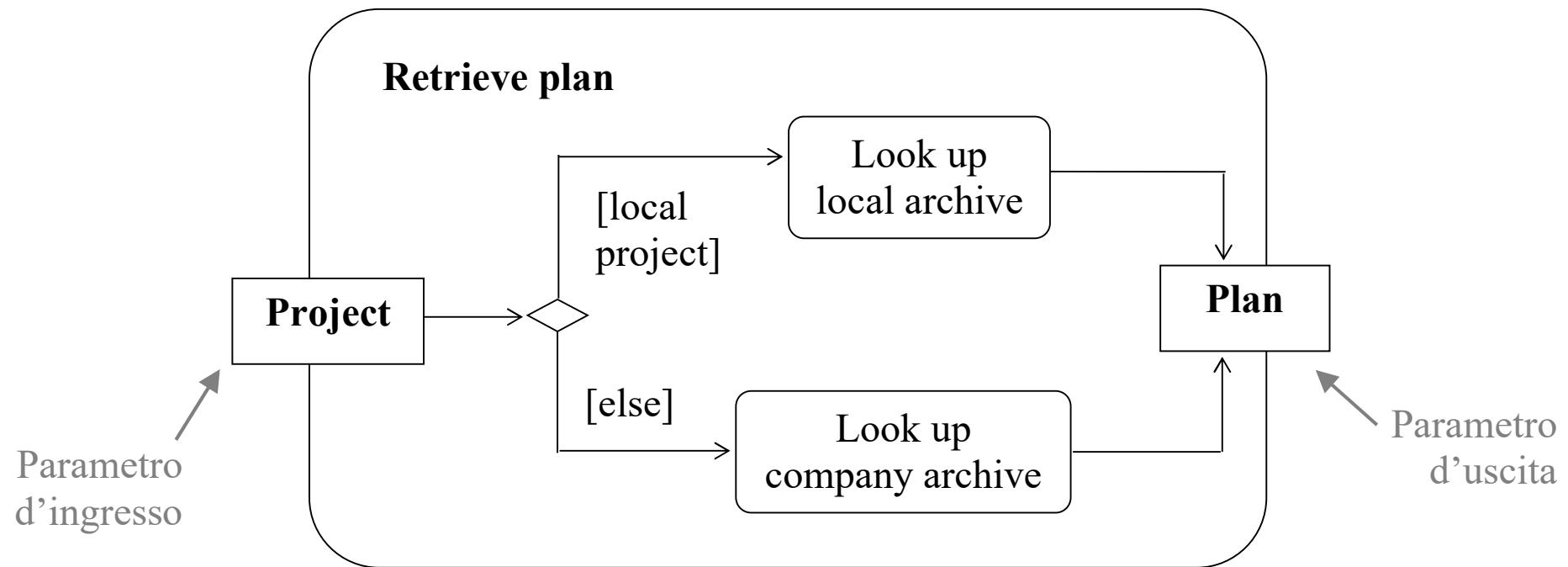


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Sotto-diagramma di attività (o diagramma di attività secondario)	<ul style="list-style-type: none">• Diagramma di attività, racchiuso nel riquadro di un'azione,• provvisto di <i>nome</i>, scritto in grassetto entro tale riquadro,• può essere dotato di parametro di ingresso e parametro di uscita	Illustra la scomposizione di un'azione di livello di astrazione superiore
Parametro di ingresso/uscita di un sotto-diagramma di attività	<p>Rettangolo</p> <ul style="list-style-type: none">• sovrapposto al riquadro del sottodiagramma di attività• contenente il <i>nome</i> di un parametro, scritto in grassetto	

Sotto-diagramma di attività



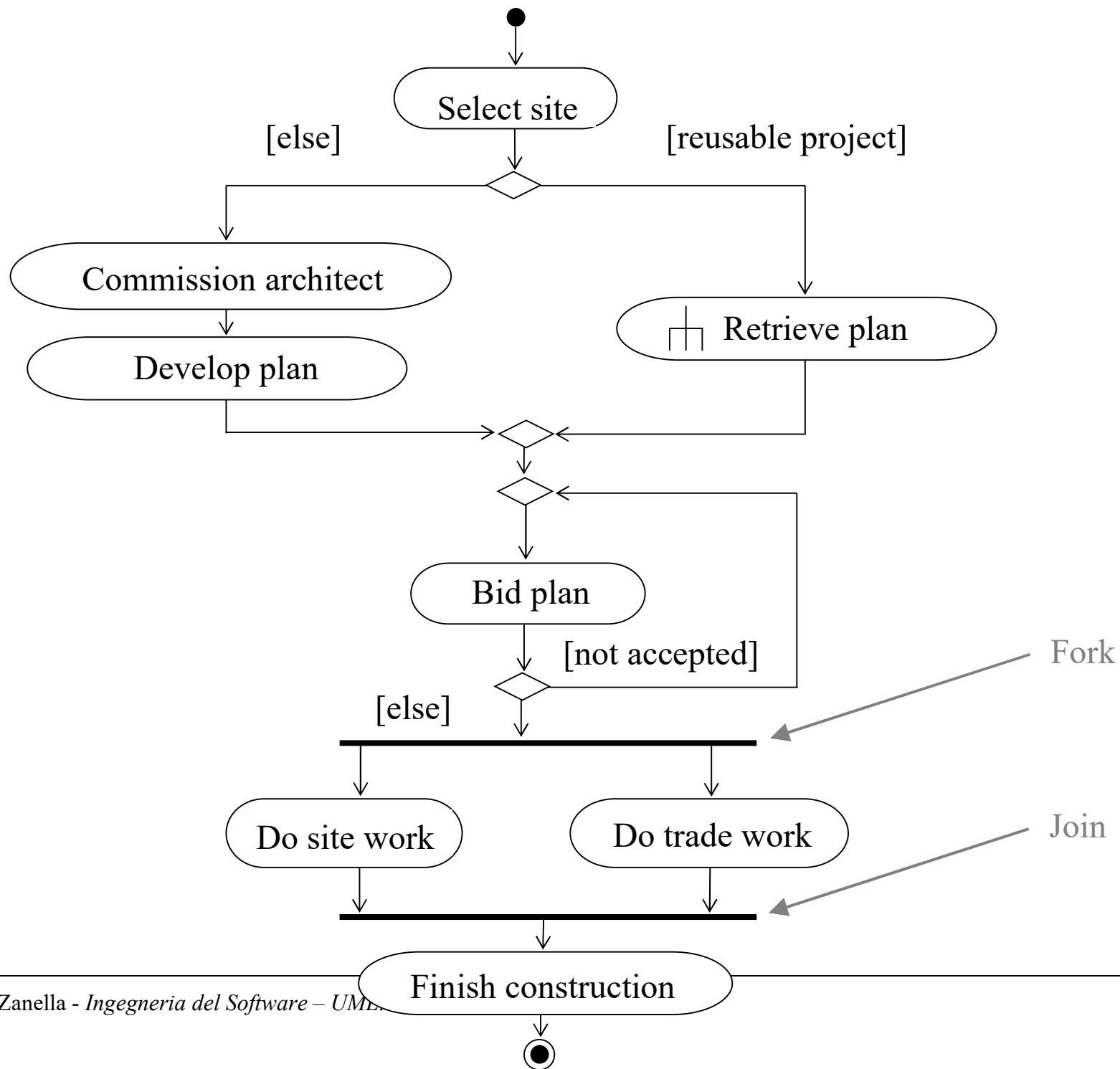
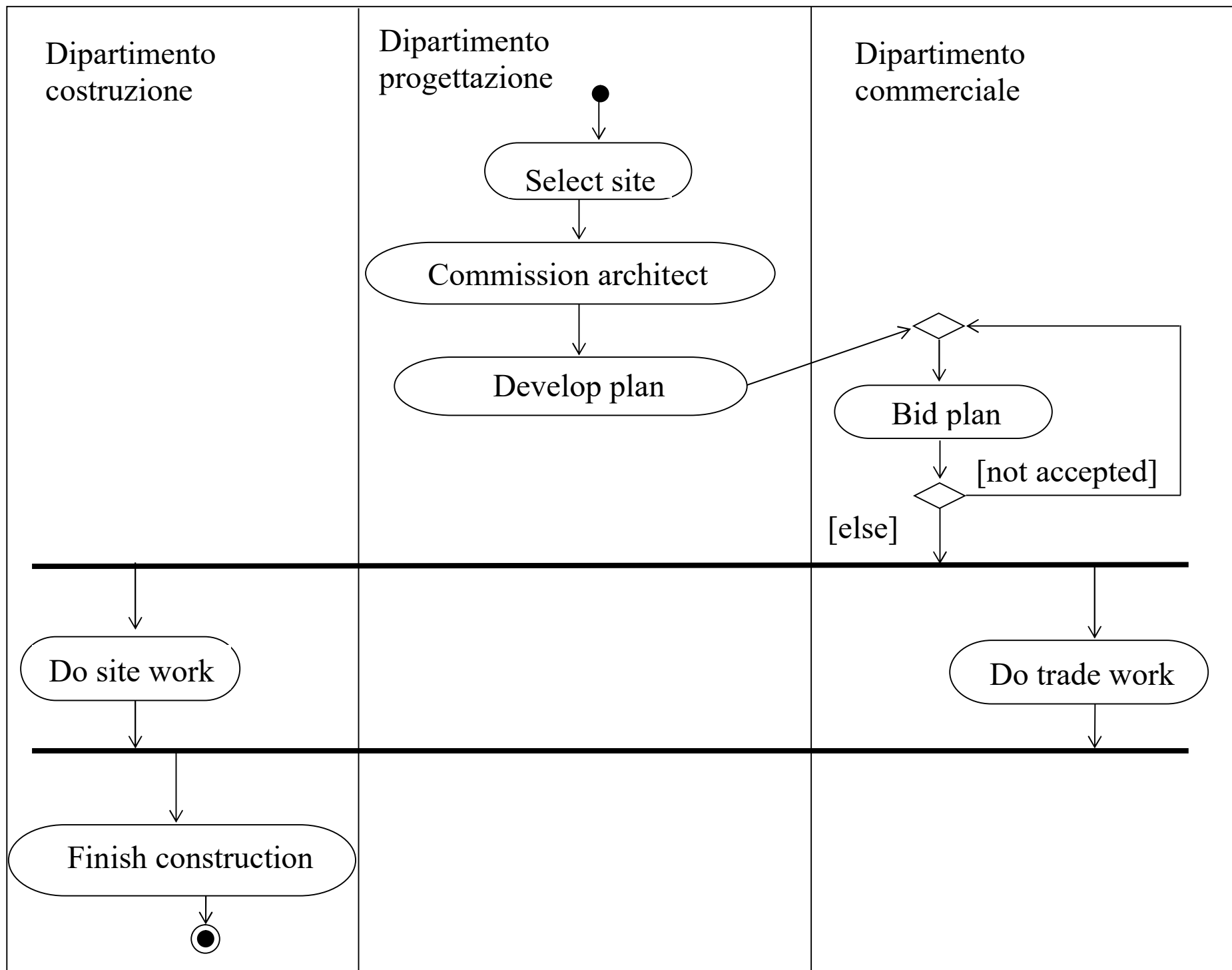


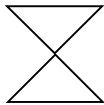
Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Partizioni	Uso di corsie (anche non lineari) o di griglie per etichettare una o più azioni con la classe responsabile (modellazione in termini di programmazione) o con le persone/dipartimenti responsabili delle attività stesse (modellazione di dominio o modellazione di processi di business)	Organizzazione dei diagrammi delle attività in responsabilità (come nei diagrammi di interazione) Se il diagramma viene suddiviso in più corsie, queste si chiamano swimlanes


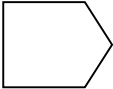


Segnali

- Un segnale dotato solo di freccia uscente rappresenta un evento di cui un'attività (o, meglio, un'azione della stessa) è costantemente in ascolto
- Se il segnale ha anche un flusso entrante, il sistema non si metterà in ascolto dell'evento finché il segnale non sarà stato attivato da tale flusso

Elementi	Sintassi	Semantica
Segnale temporale	Icona  <ul style="list-style-type: none">• accompagnata da una stringa che contiene info circa il segnale• da cui esce necessariamente un flusso e in cui può entrare un flusso	Evento, proveniente da un processo esterno, che si verifica in virtù del trascorrere del tempo

Segnali

Elementi	Sintassi	Semantica
Segnale di accettazione	Icona  <ul style="list-style-type: none">• contenente il nome del segnale• da cui esce necessariamente un flusso e in cui può entrare un flusso	Evento proveniente da un processo esterno che si verifica quando una certa condizione diventa vera
Segnale inviato	Icona  <ul style="list-style-type: none">• contenente il nome del segnale• in cui entra necessariamente un flusso e da cui può uscire un flusso	Evento generato dal processo corrente Utile quando si deve inviare un segnale e poi attendere la risposta prima di continuare

Segnali

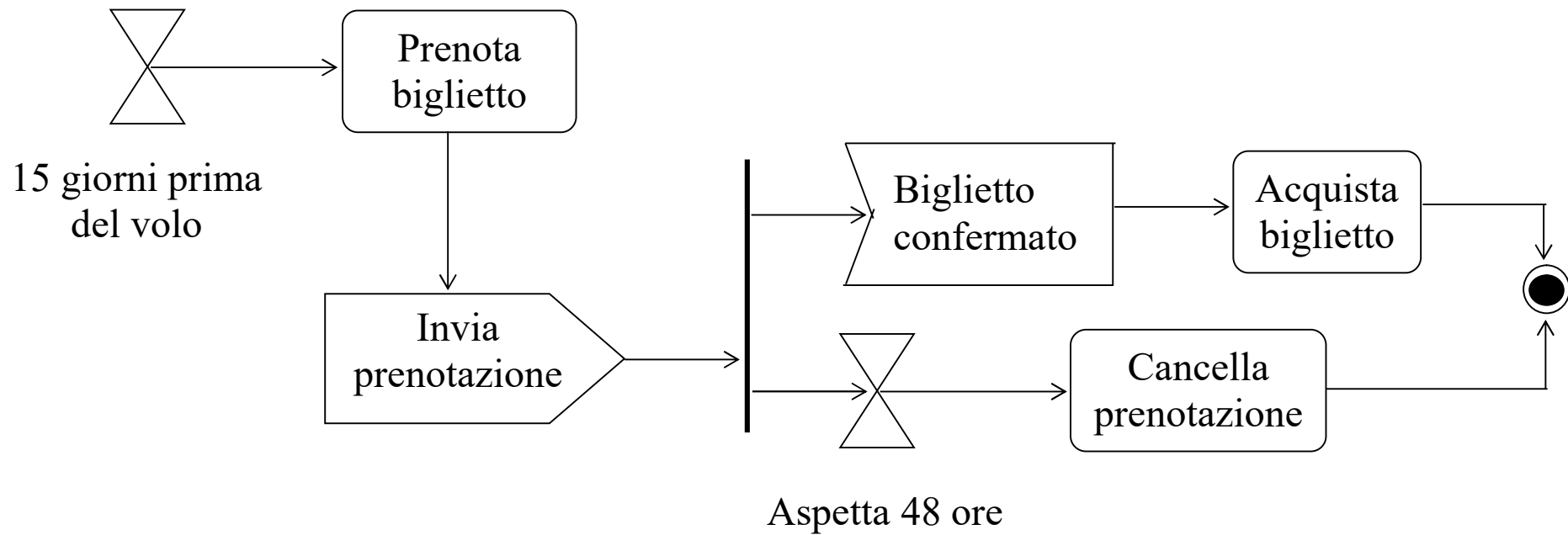


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Passaggio di un oggetto fra azioni	Scatola rettangolare <ul style="list-style-type: none">• contenente il nome di una classe• destinataria del flusso proveniente da un'azione• sorgente di un flusso destinato a un'altra azione	

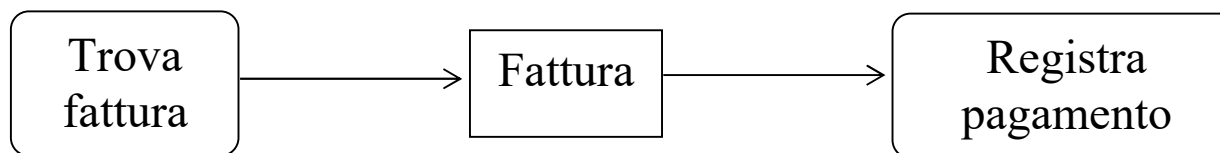


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Pin di parametro	Quadratinio posto sulla coda e sulla punta di una freccia (arco), accompagnato da una stringa (nome del parametro)	Novità di UML 2 Rappresenta un parametro passato dall'azione sorgente all'azione destinazione (il pin sulla coda è un parametro di uscita, quello sulla punta è un parametro di ingresso) Il nome del parametro è lo stesso a entrambe le estremità della freccia

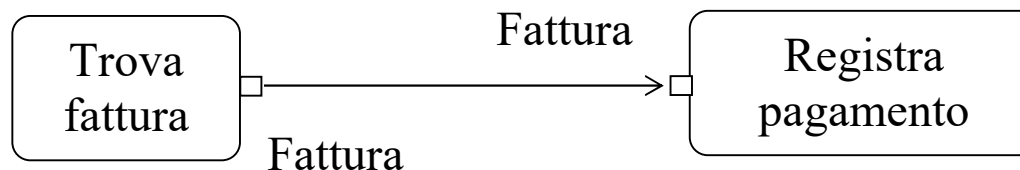


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Pin di parametro con trasformazione	<p>Quadrato posto sulla coda (unica) e sulle punte (una o più) di una freccia (arco)</p> <p>Tutte le punte della freccia devono essere dirette alla stessa azione (join implicito)</p> <p>Ciascun quadrato associato alla freccia è accompagnato da una stringa (nome del parametro) diversa da quella degli altri</p>	<p>Se un parametro di ingresso differisce dal parametro di uscita, significa che esso è ottenibile attraverso una trasformazione (priva di effetti collaterali, essenzialmente una query) del parametro di uscita</p> <p>È bene fare corrispondere a ogni trasformazione una nota contenente la parola chiave «<code>transformation</code>» e info sulla stessa</p> <p>Nella modellazione dei processi di business, i pin indicano le risorse prodotte e quelle consumate dalle azioni</p>

Pin

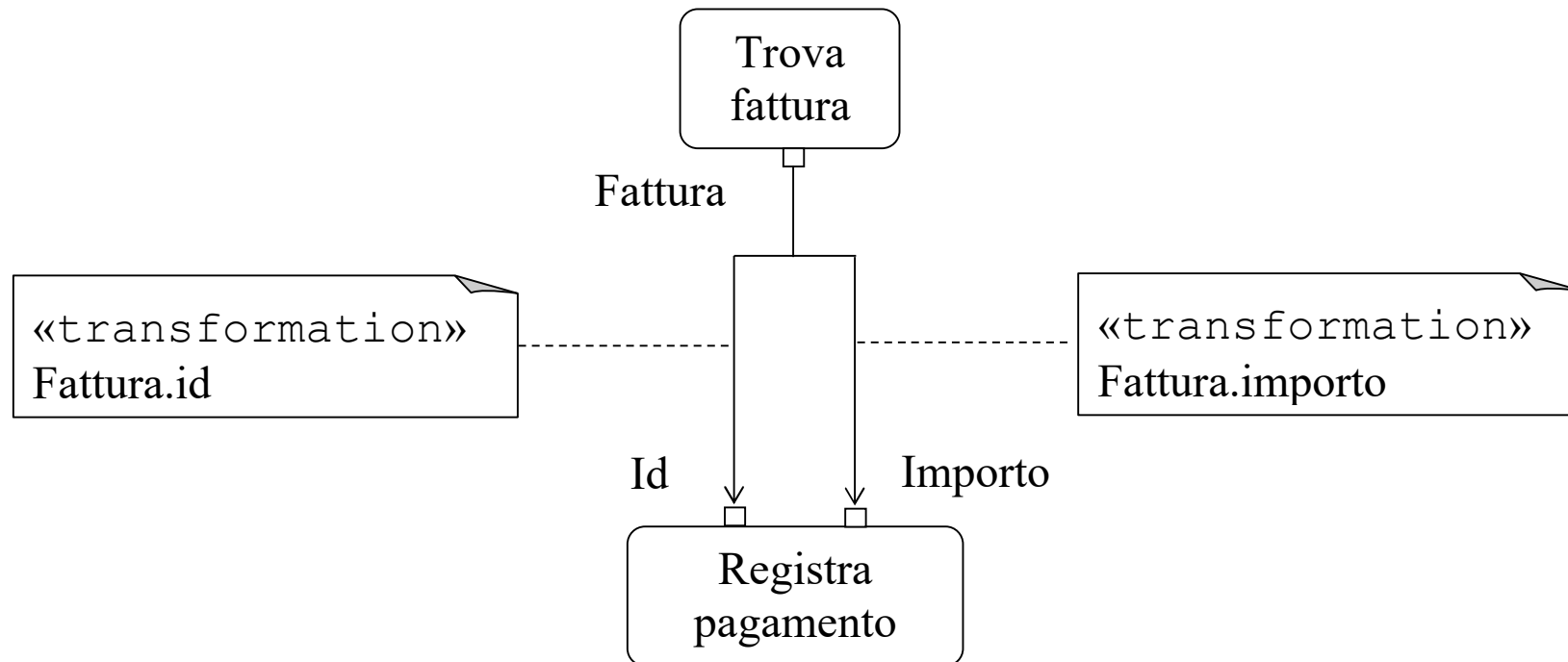


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Regione di espansione	Area di un diagramma di attività circondata da un rettangolo ad angoli smussati in linea tratteggiata, che può contenere la parola chiave «concurrent»	Le azioni contenute nell'area avvengono una volta per ogni elemento di una collezione (in parallelo per tutti gli elementi se è presente la parola chiave «concurrent»)

Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Liste di ingresso e uscita	<p>Ciascuna lista è una sequenza di 4 pin contigui</p> <ul style="list-style-type: none"> • opzionalmente dotata di nome • disegnata sovrapposta al confine di una regione di espansione; deve esistere un flusso uscente dalla lista di ingresso e diretto a un'azione interna e un flusso diretto da un'azione interna alla lista di uscita; oppure • disegnata attaccata al confine di un'azione <p>Sia A l'azione o regione di espansione a cui una lista si riferisce; allora</p> <ul style="list-style-type: none"> • se la lista è di ingresso, essa è dotata di un flusso entrante, proveniente dall'esterno rispetto ad A • se la lista è di uscita, essa è dotata di un flusso uscente, diretto all'esterno di A 	<p>Una lista rappresenta una collezione di elementi in ingresso o in uscita a una regione di espansione o a un'azione</p> <p>Un'azione dotata di liste di ingresso e uscita viene eseguita in modo concorrente sui vari elementi della collezione in ingresso</p>

Liste di ingresso e uscita

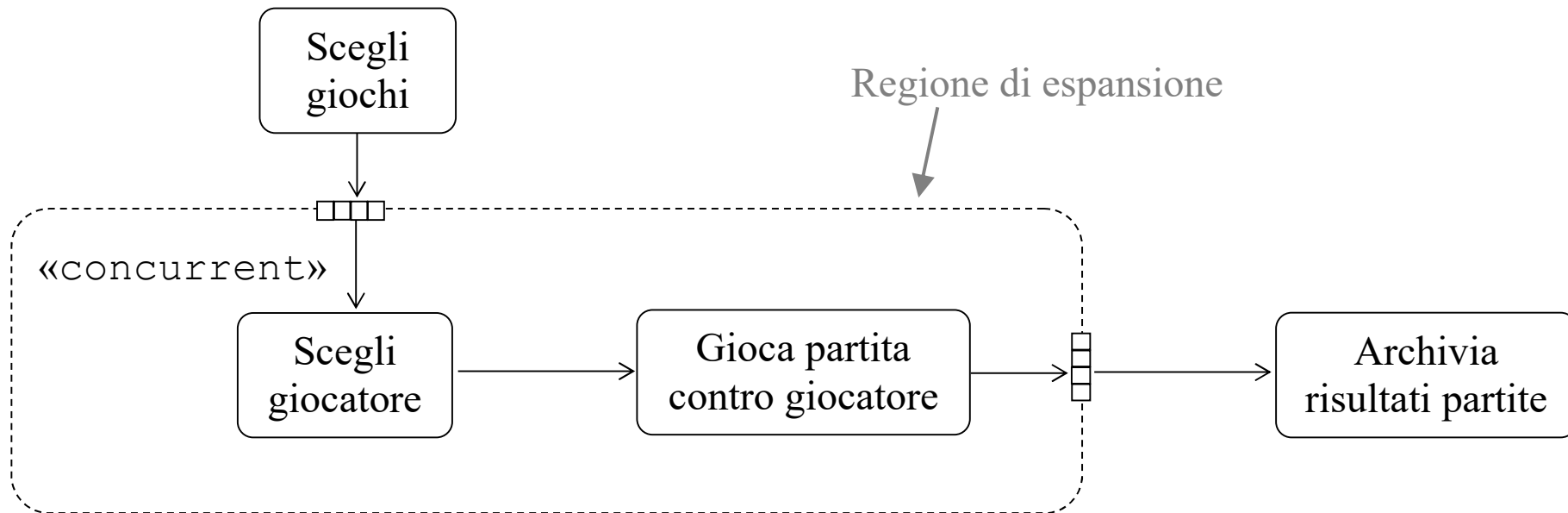
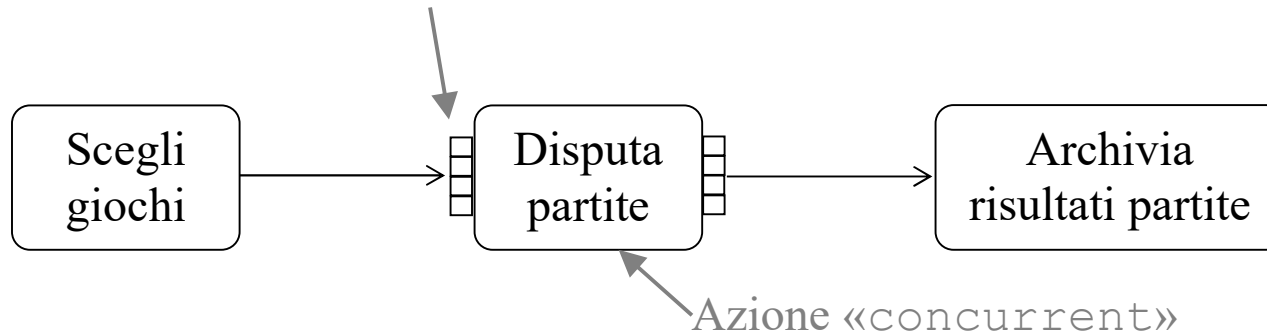



Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Fine flusso (<i>flow final</i>)	Icona  dotata di un flusso entrante e di nessun flusso uscente Può essere contenuta entro una regione di espansione	Indica la fine di un particolare flusso di esecuzione, senza che per questo abbia termine l'intera attività Consentendo al flusso di esecuzione contenuto in una regione di espansione di terminare selettivamente per alcuni elementi di ingresso, permette alle regioni di espansione di funzionare come filtro, producendo in uscita una collezione più piccola di quella di ingresso

Regione di espansione come filtro

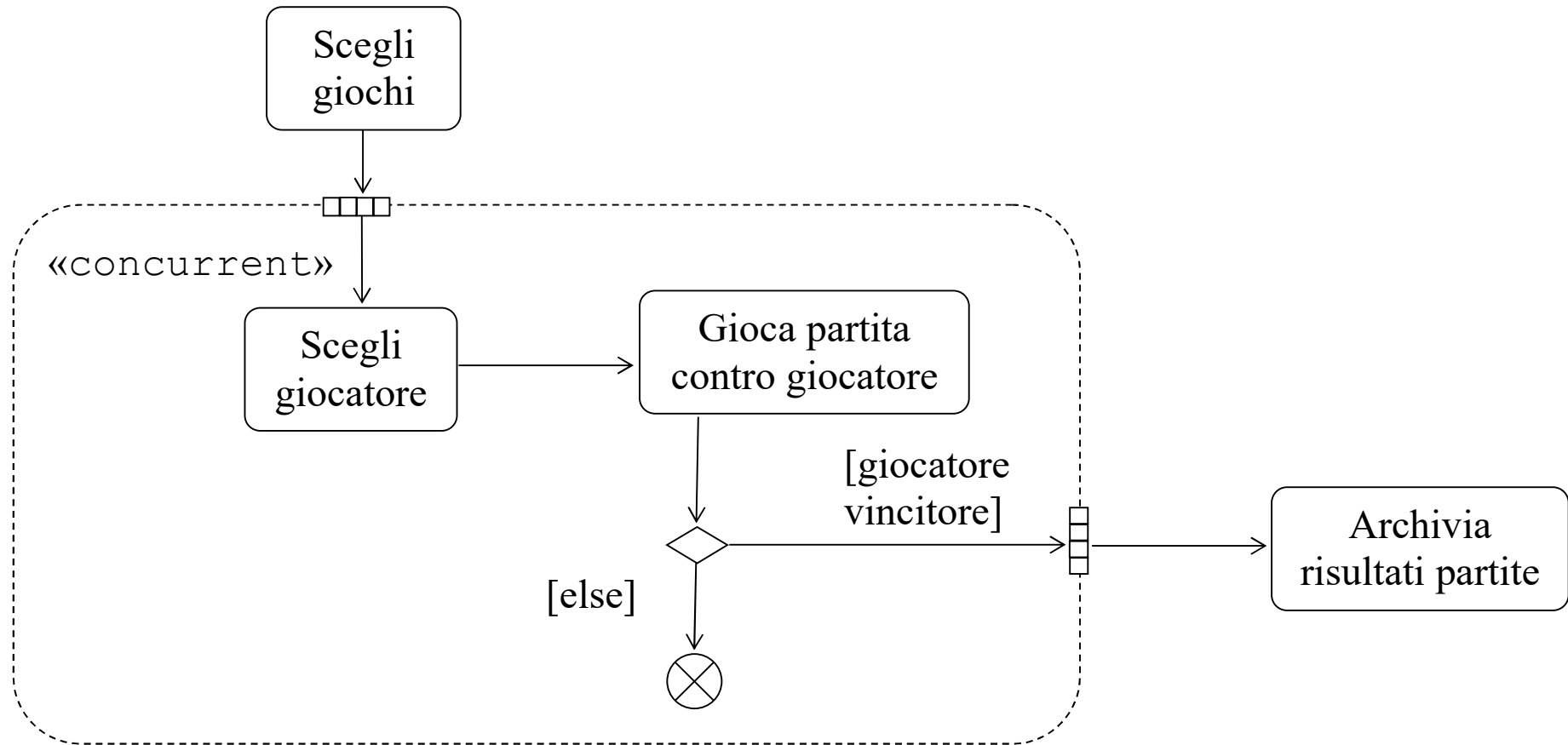
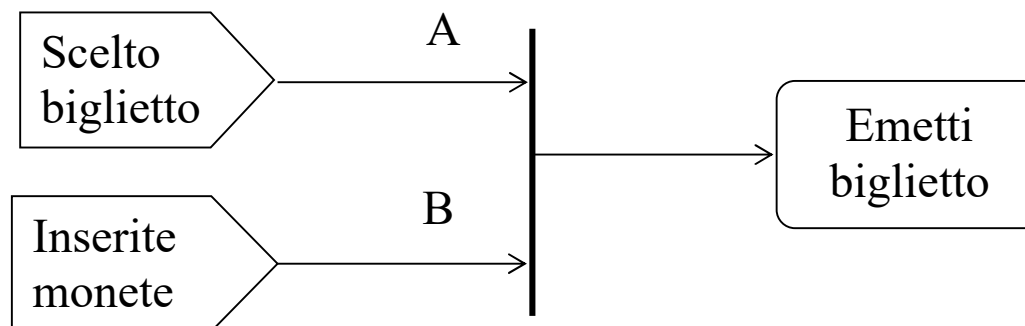


Diagramma di attività (cont.)

Elementi	Sintassi	Semantica
Specifica di join	<p>Vincolo $\{joinSpec = espressione_booleana\}$ associato a un join</p> <p>I flussi entranti nel join possono essere etichettati</p>	<p>Ogni volta che un flusso entrante arriva al join, l'espressione viene valutata: se risulta vera, il controllo passa al flusso di uscita</p> <p>Per indicare nell'espressione booleana che il join ha ricevuto un flusso, si può usare l'etichetta di tale flusso come variabile logica</p>



$\{ joinSpec = A \text{ and } B \text{ and } \text{valore delle monete inserite} \geq \text{prezzo del biglietto selezionato} \}$

Diagrammi di attività: a cosa servono?

- Portano a identificare le sequenze non necessarie dei business e a sfruttare il parallelismo, aumentando così l'efficienza e la reattività dei processi
- Nella modellazione di business, incoraggiano un esperto del dominio a trovare nuovi modi per fare le cose
- Consentono di raffigurare i thread di programmi concorrenti e i punti nei quali è necessario sincronizzarli
- Consentono di esplorare il comportamento del sistema in corrispondenza dei casi d'uso senza assegnare immediatamente le responsabilità alle varie classi
- Documentano i metodi

Qualità del Software

La qualità è relativa

Prospettive sulla qualità	Osservatore
<i>Trascendentale</i> : la qualità è qualcosa che possiamo riconoscere ma non definire (ideale verso cui tendere)	filosofo
<i>Utente (qualità esterna)</i> : la qualità è appropriatezza rispetto allo scopo	clienti, operatori commerciali
<i>Produzione</i> : la qualità è conformità alle specifiche (del processo costruttivo)	sviluppatori
<i>Prodotto (qualità interna)</i> : la qualità deriva dalle caratteristiche intrinseche del prodotto (metriche del sw)	ricercatori
<i>Valore</i> : la qualità dipende da quanto il cliente è disposto a pagarla	imprenditore

Qualità del sw: una duplice classificazione

Il nocciolo dell'ingegneria del software è l'attenzione alla qualità (Pressman, 2004)

Primo criterio:

- Qualità di prodotto
- Qualità di processo (influenza quella di prodotto)

Secondo criterio:

Qualità esterne: percepibili da un osservatore esterno che esamina una black-box (sono quelle di interesse per SE)

Qualità interne: percepibili esaminando la struttura interna di una white-box (sono quelle che permettono di realizzare le qualità esterne)

Fattori di qualità

Di prodotto

- Correttezza
- Affidabilità
- Robustezza
- Sicurezza
- Innocuità
- Prestazioni
- Usabilità
- Portabilità
- Interoperabilità

Sia di prodotto sia di processo

- Verificabilità
- Manutenibilità
- Riutilizzabilità
- Comprensibilità

Di processo

- Produttività
- Visibilità
- Tempestività

Correttezza

- Il sw è corretto se soddisfa le specifiche dei requisiti funzionali (caratteristica oggettiva)
- Se tali specifiche sono formali, la correttezza può essere provata formalmente (mediante una dimostrazione di teorema) dal momento che i programmi sono oggetti formali, oppure smentita mediante controesempi (attività di testing)

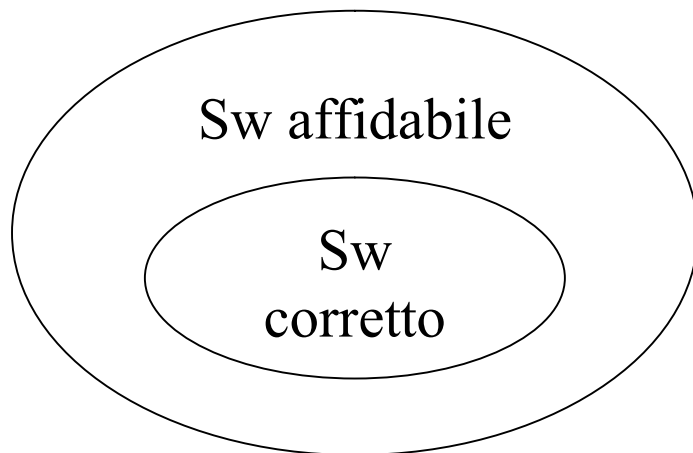
Limiti:

- È una qualità assoluta (sì/no), non esiste alcun concetto di “grado di correttezza” né di gravità dell’infrazione
- E se le specifiche fossero sbagliate (magari a causa di requisiti inattendibili o di errori nella conoscenza di dominio)?

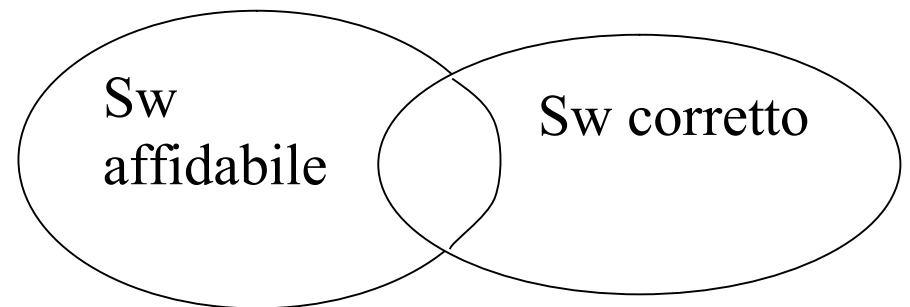
Affidabilità

- Informalmente significa che l'utente si può fidare del prodotto sw in questione
- I risultati delle elaborazioni sono quelli voluti o presentano disturbi tollerabili (concetto soggettivo e variabile da sistema a sistema)
- Matematicamente è definibile come la “probabilità di assenza di malfunzionamenti nell'unità di tempo”
- Se le specifiche sono corrette (situazione ideale), un sistema sw corretto è anche affidabile ma non viceversa

Situazione ideale



Situazione reale



Robustezza

Il prodotto sw si comporta in modo ragionevole anche in circostanze non previste dai requisiti (valori d'ingresso non validi, malfunzionamenti hw, ecc.)

Sicurezza (*security*)

Capacità del sistema sw in esecuzione di impedire l'accesso a info private

Innocuità (*safety*)

- Assenza di pericolosità (o tollerabilità della pericolosità) dell'elaborazione
- Capacità di operare senza malfunzionamenti catastrofici

Prestazioni

- Tempi di elaborazione, occupazione di memoria, traffico in rete
- Dipendono da un uso efficiente delle risorse (memoria, tempo di elaborazione, canali di comunicazione)
- Influenzano l'usabilità e la “scalabilità” di un'applicazione (una soluzione che funziona su una LAN magari non funziona su una WAN)
- Possono essere verificate mediante
 - ✓ Analisi di complessità (insegnamento di “Algoritmi e Strutture Dati”)
 - ✓ Misure durante esecuzioni-campione
 - ✓ Simulazione di un modello
 - ✓ Analisi di un modello (es. analisi probabilistica secondo la teoria delle code)

Usabilità

- Intuitività, naturalezza ed ergonomia dell'interazione col sistema sw dal punto di vista delle diverse categorie di utenti dello stesso (espressione vecchia: user-friendliness)
- È influenzata dalla tipologia di interfaccia utente (ad es. testuale o grafica)
- Dipende da coerenza, prevedibilità, gradevolezza ed esplicatività dell'interfaccia
- L'utente non deve provare una sensazione di smarrimento in alcuna occasione

Portabilità

- Un prodotto sw è portabile se può essere eseguito su più piattaforme hw e/o sw
- È una proprietà importante anche quando vengono introdotte nuove piattaforme e nuovi ambienti o quando la rete aziendale è eterogenea

Interoperabilità

Capacità di un sistema sw di coesistere e cooperare con altri sistemi o con servizi web (attraverso middleware)

Verificabilità

- Facilità di verifica di altre proprietà del prodotto (ad es. correttezza o prestazioni)
- È prevalentemente una qualità interna ma può essere anche esterna (ad es. verificabilità della “sicurezza”)
- Lo stato di avanzamento di un progetto deve essere controllabile, unitamente al grado di soddisfacimento dei vincoli

Manutenibilità

Facilità e rapidità con cui è eseguibile la manutenzione

Manutenzione = attività post-rilascio (50-70% dei costi complessivi); si articola in:

- Correttiva ($\cong 20\%$ dei costi di manutenzione): correzione dei difetti residui
- Adattativa (20-25%): adattamento a cambiamenti nell'ambiente
- Perfettiva (50%, è conseguenza della malleabilità del sw): aggiunta/rimozione di funzionalità, miglioramento di alcune caratteristiche di qualità → differenza del concetto di manutenzione rispetto all'ingegneria tradizionale

Situazione di retroazione: l'organizzazione del lavoro fa nascere esigenze di automazione mediante sw → introduzione del sw e conseguente modifica dell'organizzazione del lavoro → necessità di modifiche del sw

Manutenzione del processo produttivo: modifiche ai piani inizialmente previsti

Riusabilità

- Facilità e rapidità con cui prodotti/componenti/processi esistenti possono essere riutilizzati, dopo lievi modifiche, per costruire altri prodotti
- Il riutilizzo di parti standardizzate è una misura della maturità di un settore ingegneristico

Comprensibilità

- Facilità di comprensione di un sistema sw (per modificare un sistema sw è innanzi tutto necessaria la comprensione dello stesso)
ATTENZIONE: non si intende facilità di comprensione del solo codice ma anche di altri artefatti e delle loro relazioni
- Facilità di comprensione di un processo

Produttività

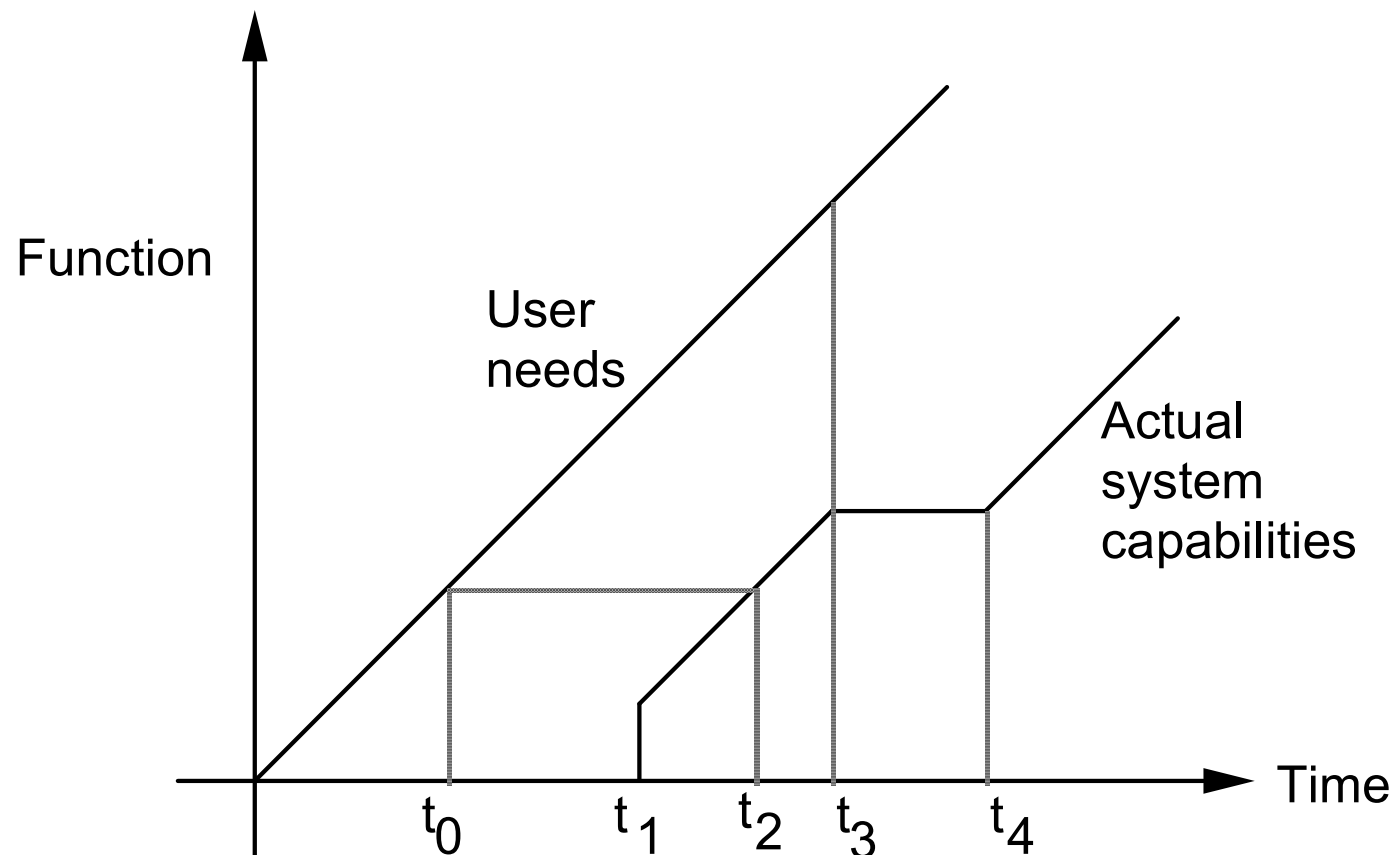
- Capacità di un processo di incrementare il gettito produttivo (ad es. mettendo a disposizione CASE tool)
- È difficile da caratterizzare

Visibilità

- Ogni passo del processo è chiaramente documentato
- In ogni istante è noto lo stato corrente

Tempestività

- Capacità di consegnare un prodotto sw con puntualità (o di presentarlo al momento giusto sul mercato)
- Spesso il processo produttivo non segue l'evoluzione dei requisiti degli utenti, sussiste bensì uno scostamento (mismatch) fra i requisiti degli utenti e lo stato del prodotto



Aree applicative del sw

Classificazione per insiemi omogenei di caratteristiche di qualità da quantificare all'avvio di un progetto

- Sistemi informativi
- Sistemi in tempo reale
- Sistemi distribuiti
- Sistemi embedded

Molti sistemi hanno caratteristiche comuni a più aree (es. sistema informativo distribuito e in tempo reale, sistema embedded in tempo reale)

Sistemi informativi

- Si interfacciano a una base di dati
- Sono orientati ai dati
- Molti forniscono una GUI web
- Molti consentono personalizzazioni (ad es. definizione e generazione di nuovi rapporti)

Es.: sistemi bancari, bibliotecari, di gestione del personale

Requisiti:

- Integrità dei dati
- Security
- Disponibilità dei dati
- Prestazioni relative alle transazioni

Sistemi in tempo reale

- Devono rispondere a determinati eventi in ingresso entro un intervallo di tempo predefinito e spesso molto limitato (quindi sussistono vincoli quantitativi relativi ai tempi di risposta)
- Spesso fanno parte di sistemi complessi (di automazione di fabbrica, sorveglianza, ecc.)
- Sono orientati al controllo

Es.: sistemi di monitoraggio di pazienti/impianti, sistemi di controllo del volo di un aereo, sistemi di difesa, sistema di gestione del mouse

N.B. Definizione sbagliata di sistema in tempo reale: sistema che richiede tempi di risposta veloci. Infatti una risposta troppo veloce può essere scorretta quanto una risposta tardiva

Sistemi distribuiti

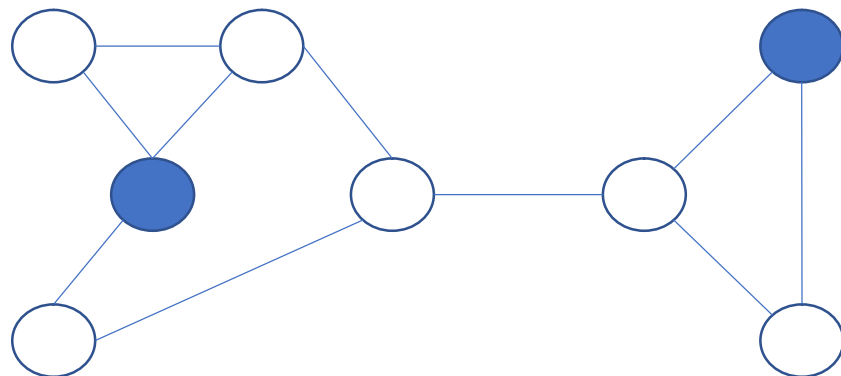
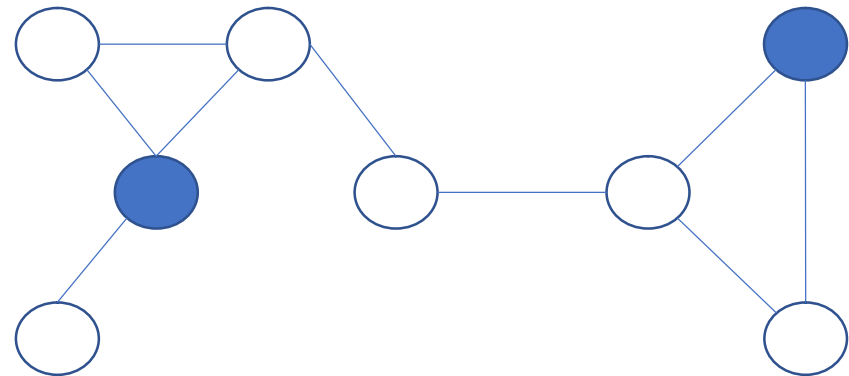
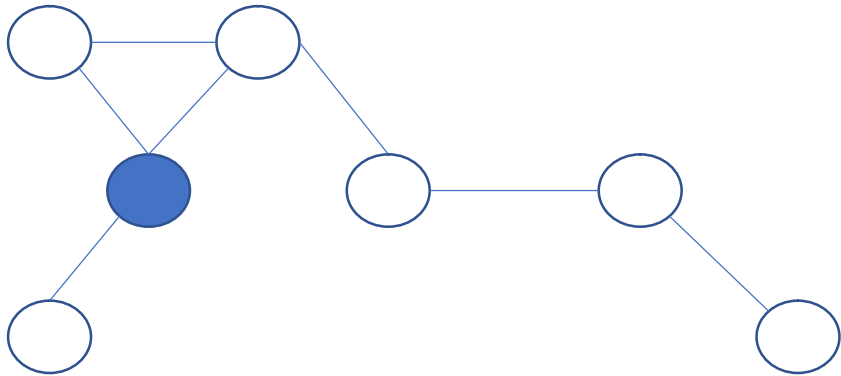
Distribuzione su computer diversi, collegati da una rete di TLC, di

- dati e/o
- componenti sw eseguibili

Requisiti da definire:

- Livello di distribuzione
- Possibilità di tollerare il partizionamento (della rete in sottoreti disgiunte)
- Tolleranza per l'indisponibilità di uno o più computer

Sistemi distribuiti: nuove strade per ottenere la qualità



Sistemi distribuiti: nuove strade per ottenere la qualità

L'eventuale replica dei dati su più macchine aumenta:

- affidabilità
- prestazioni

Java definisce un linguaggio intermedio (*bytecode*) che può essere interpretato su ogni computer → i componenti possono essere caricati in rete in maniera dinamica quando ciò risulta necessario (mobilità del codice)

L'eventuale trasferimento dinamico del codice al nodo che memorizza i dati sui quali il codice deve operare (es. applet Java) migliora le prestazioni

Sistemi embedded

- Il sw è solo uno dei componenti di questi sistemi, quello che controlla gli altri, interfacciandosi con essi
- I requisiti del sw devono essere bilanciati con quelli delle altre parti
- Spesso privi di interfacce rivolte all'utente finale

Es. sw di controllo di aerei, robot, elettrodomestici, cruscotto dell'automobile, telefoni cellulari, macchine distributrici, sistemi di commutazione telefonica