Modellare sistemi software con UML

Andrea Baruzzo
e-mail: baruzzo@dimi.uniud.it

Dipartimento di Matematica e Informatica
Università degli Studi di Udine

Agenda

- **Introduzione: approccio e motivazioni**
- Modellare la struttura statica
- Modellare la struttura dinamica

Introduzione: l'approccio

- Approccio *pratico* alla modellazione e alla verifica di sistemi software *reali*
- Conseguenza: tecnologie, strumenti e linguaggi largamente utilizzati in *ambito industriale*
- Scelte soggettive
  - Sistemi software target: sistemi object-oriented
  - Linguaggi di programmazione target: C++, Java
  - Linguaggio di modellazione: UML

Linguaggio di programmazione o di modellazione?

- Perché un linguaggio di modellazione?
  - Spostare discussioni di design dal codice al modello fornisce un più alto *livello di astrazione* indipendente dalla sintassi e dai dettagli del singolo linguaggio di programmazione
  - Un buon design è caratterizzato da alcuni indici di qualità come la stratificazione, la distribuzione del controllo, l'incapsulamento, la coesione
  - Questi indici di qualità si vedono meglio su un modello grafico che sulle righe di codice

Introduzione: Perché UML per fare verifica?

- Perché UML?
  - E' uno *standard de facto* a livello industriale per descrivere sistemi software object-oriented
  - Esiste una *semantica formale* per almeno una parte del linguaggio (classi, stati)

- Unified Modeling Language: *chi, dove, quando*
  - Booch, Jacobson e Rumbaugh; (UML 0.9)
  - Standard OMG (Object Management Group) dal 1997
  - Versione attuale: 2.0 (06/2005)

- Diagrammi per descrivere diversi aspetti di un sistema software:
  - struttura statica (architettura, sottosistemi, moduli, …)
  - struttura dinamica (scenari d'esecuzione, interazioni, stati, …)
  - requisiti funzionali (casi d'uso)
  - vincoli (temporali, performance, contratti software)

UML: Bibliografia ragionata

- **The Unified Modeling Language User Guide 2/E**
  by Grady Booch, James Rumbaugh, Ivar Jacobson - Addison-Wesley Professional, 2005
- **The Unified Modeling Language Reference Manual 2/E**
  by James Rumbaugh, Ivar Jacobson, Grady Booch - Addison-Wesley Professional, 2004
- **UML Distilled: A Brief Guide to the Standard Object Modeling Language 2/E**
  by Martin Fowler - Addison-Wesley Professional, 2003
- **Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development 3/E**
  by Craig Larman - Addison-Wesley Professional, 2004
- **UML 2 and the Unified Process : Practical Object-Oriented Analysis and Design 2/E**
  by Jim Arlow, Ila Neustadt - Addison-Wesley Professional, 2005
- **UML Bible**
  by Tom Pender - Wiley, 2003

Parte I: Modellare sistemi software con UML

- Introduzione: approccio e motivazioni
- **Modellare la struttura**
  - Diagrammi di classe
  - Diagrammi degli oggetti
  - Diagrammi dei componenti
  - Diagrammi di deployment
  - Diagrammi delle strutture composte (composite structure) – (UML 2.0)
- Modellare la dinamica

Modellare la struttura

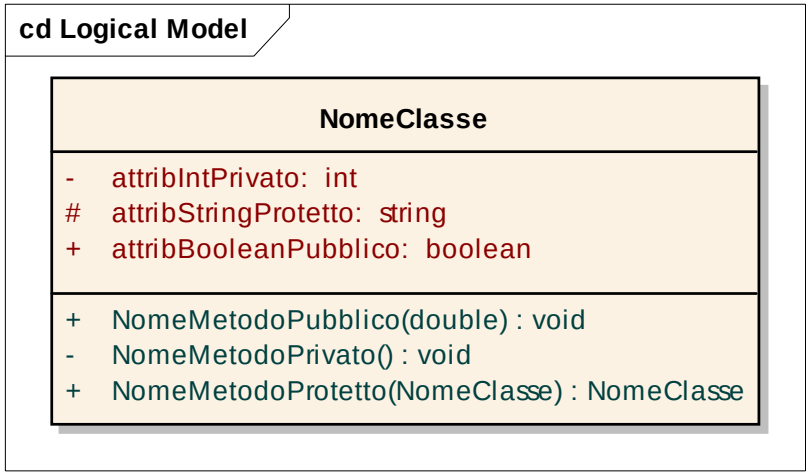
- Struttura come sinonimo di diverse cose
  - *architettura* (modularità, gestione del controllo, …)
  - *organizzazione logica* (sottosistemi, interfacce,…)
  - *organizzazione fisica* (librerie, componenti,…)
- Descrive una *vista* (prospettiva) del sistema che pone l'accento sulla *struttura statica* del sistema (classi, relazioni, attributi, operazioni, gerarchie, componenti, …)
- Quali diagrammi?
  - diagrammi di classe e degli oggetti
  - diagrammi dei componenti
  - diagrammi di deployment
  - diagrammi delle strutture composte (composite structure)

Diagrammi di classe

- Classi, attributi, metodi
- Relazioni fra classi
- Ereditarietà
- Contenimento
- Associazione, dipendenze
- Classi astratte, interfacce
- Package

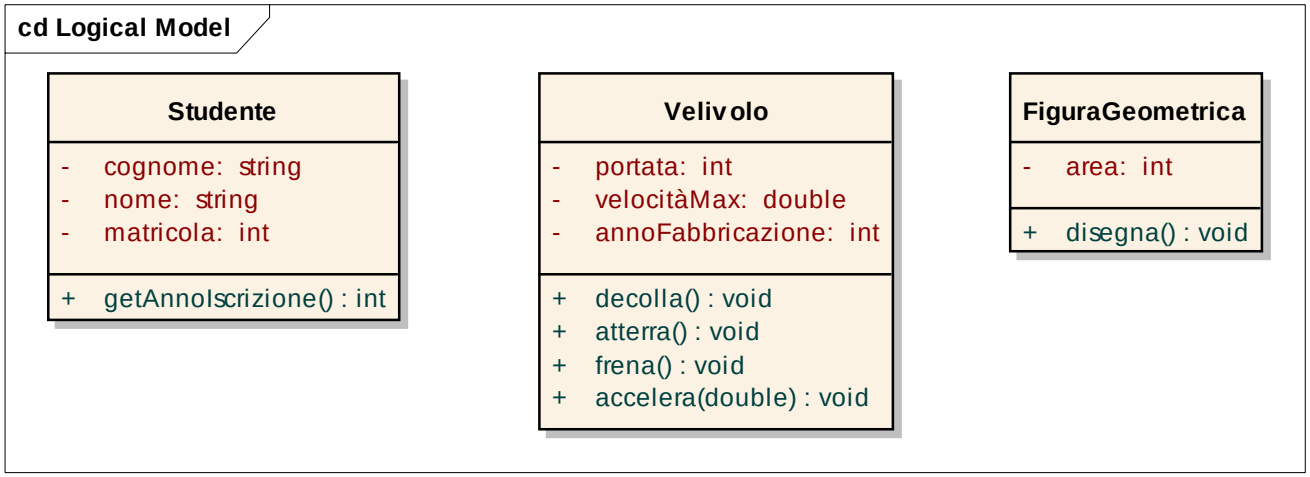
Classi, attributi, metodi

- Rettangolo con tre compartimenti:
  - Nome della classe
  - Attributi (nome, tipo)
  - Metodi (nome, args, tipo ritorno)



- Visibilità:
  - + pubblica (il default per i metodi)
  - - privata (tipica per gli attributi)
  - # protetta (ereditarietà protetta, visibilità legata al package in Java)
- Non è obbligatorio mostrare tutti i compartimenti

Alcuni esempi



- Convenzioni sui nomi
  - Classe: lettera maiuscola iniziale per ogni parola
  - Attributi: lettera minuscola iniziale,
    maiuscola iniziale per ogni successiva parola
  - Metodi: (C++) maiuscola iniziale per ogni parola
    (Java) stessa degli attributi

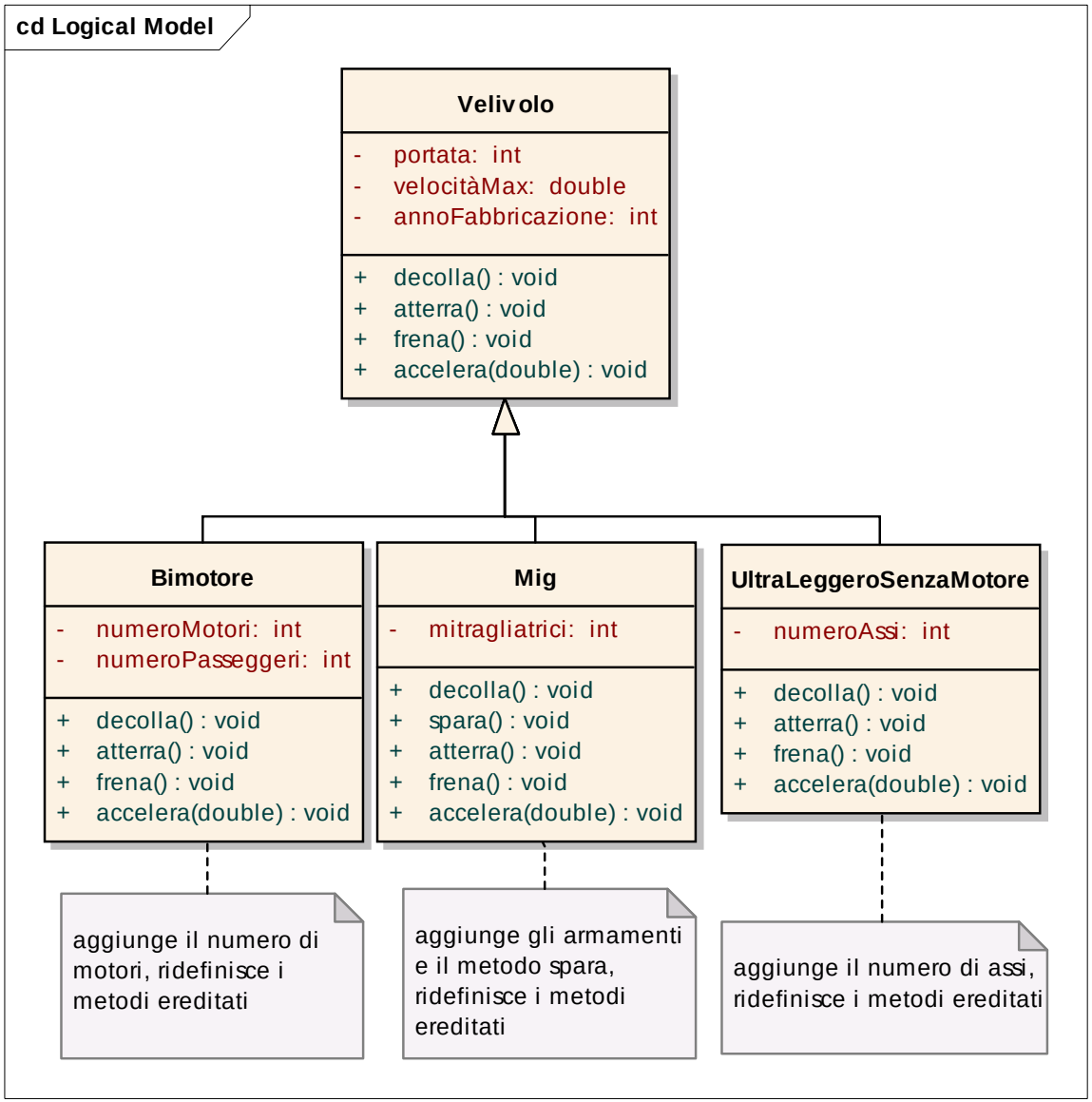
Relazioni fra classi

- Relazioni di:
  - Ereditarietà
  - Contenimento
  - Associazione
  - Dipendenze
  - Classi astratte, interfacce
  - Package

Ereditarietà

- Classe base, classi derivate
- La classe base definisce attributi comuni (ereditati)
- La classe base definisce interfaccia comune (ereditata)
- Le classi derivate specializzano la classe base aggiungendo attributi e/o comportamenti (metodi)
- Le classi derivate possono anche ridefinire metodi ereditati
- Le classi derivate possono invocare i metodi delle classi base, purché definiti pubblici o protetti

Esempio di gerarchia di classi

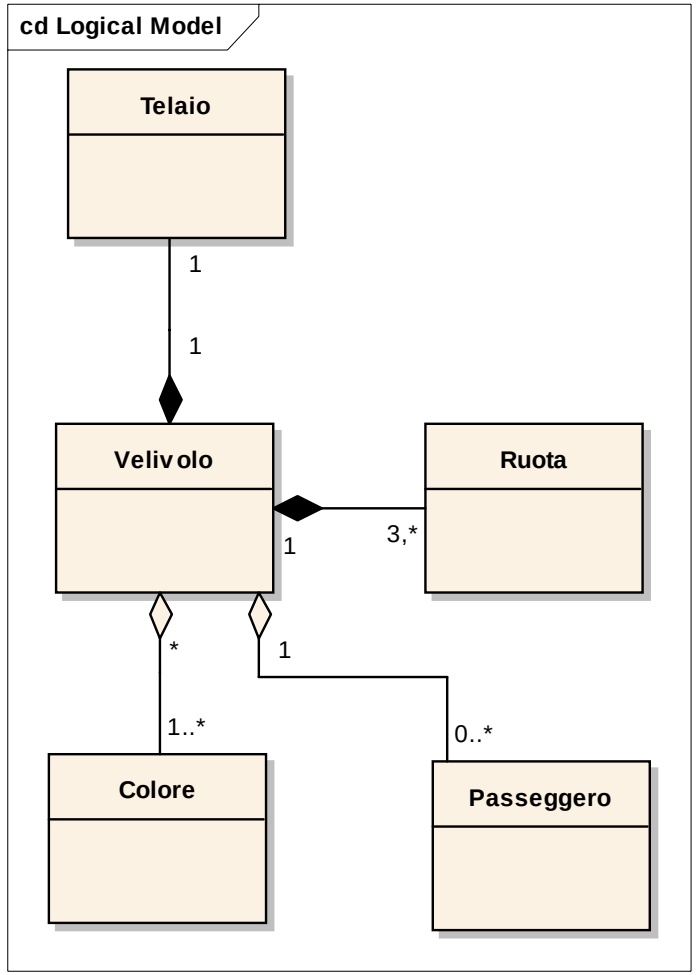


Contenimento

- Due forme:
  - Forte, detta composizione
  - Debole, detta aggregazione
- La composizione associa composto e componente per tutta la vita dei due oggetti (è una *relazione permanente*)
- La composizione è una *relazione esclusiva*: uno specifico oggetto componente non può appartenere a due composti contemporaneamente
- Se questi vincoli non sono soddisfatti, si usa l'aggregazione

Esempi di contenimento (composizione + aggregazione)

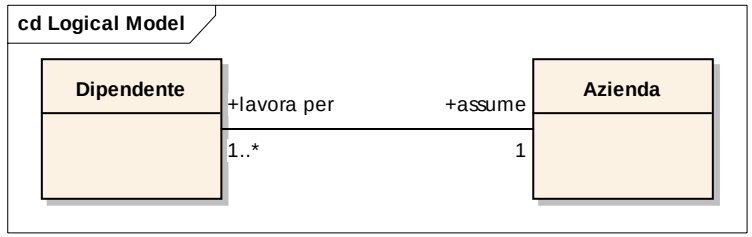
- Composizione (rombo pieno) e aggregazione (rombo vuoto)
- Le cardinalità sono importanti!
- L'aggregazione è sinonimo di (possibile) condivisione; semantica del contenimento "by reference" (puntatori)
- L'aggregazione è una relazione temporanea e non esclusiva

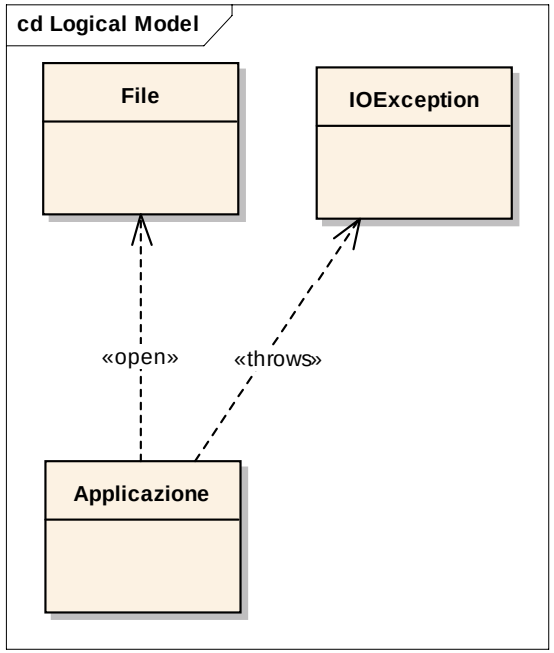


Associazione e dipendenze

- L'associazione esprime un legame *permanente* che vale per tutta la vita degli oggetti coinvolti
- Evidenzia anche i ruoli
- La dipendenza esprime invece un generico legame *temporaneo* (ad es. una chiamata)
- Lo «stereotipo» chiarisce la semantica della ipendenza

Associazione



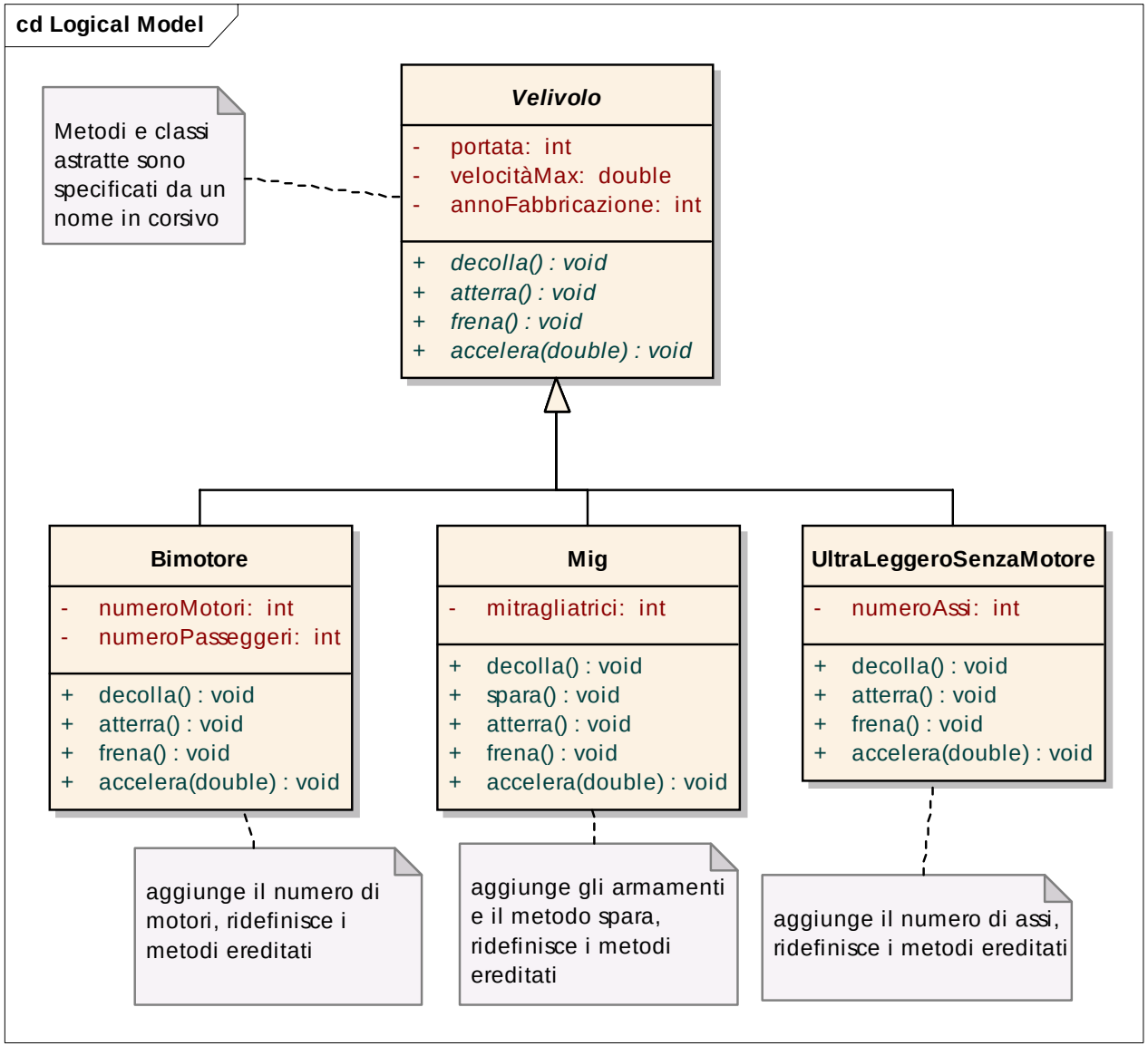


Dipendenza

Classi astratte e interfacce

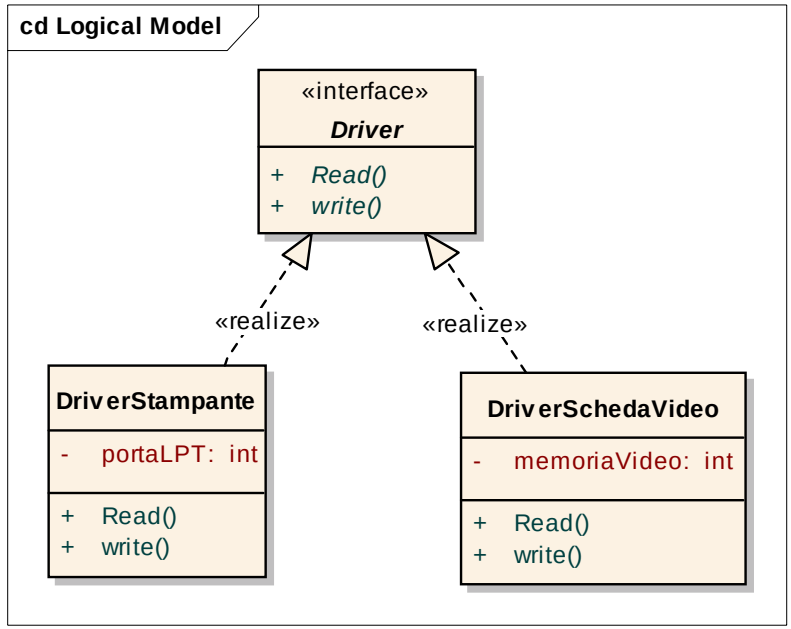
- Una classe astratta ha almeno un metodo virtuale puro (metodo astratto puro)
- Un'interfaccia non ha attributi e ha solo metodi virtuali puri (metodi astratti puri)
  - Eccezione Java: le interfacce possono dichiarare come attributi delle costanti
- Le classi che estendono (derivano da) una classe base astratta devono fornire un'implementazione per tutti i metodi astratti
- Lo stesso vale nel caso delle interfacce

Esempio di classe astratta



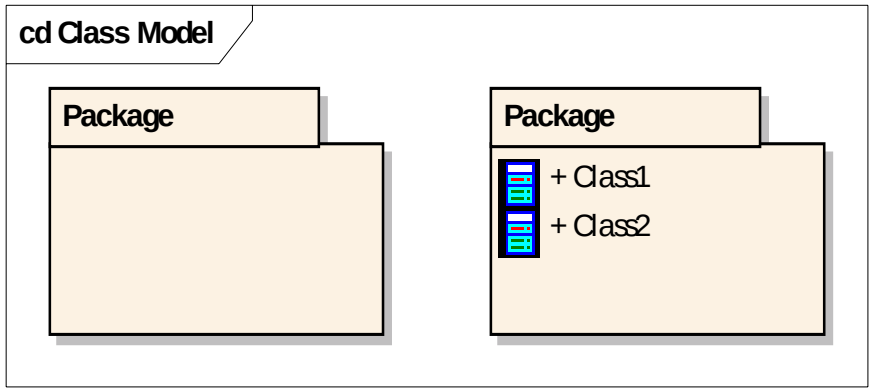
Esempio di interfaccia

- La freccia tratteggiata indica esattamente la "*realizzazione*",ossia l'implementazione di un'interfaccia
- Da non confondere con la freccia continua dell'ereditarietà
- Lo stereotipo chiarisce la semantica, ma è opzionale



Package

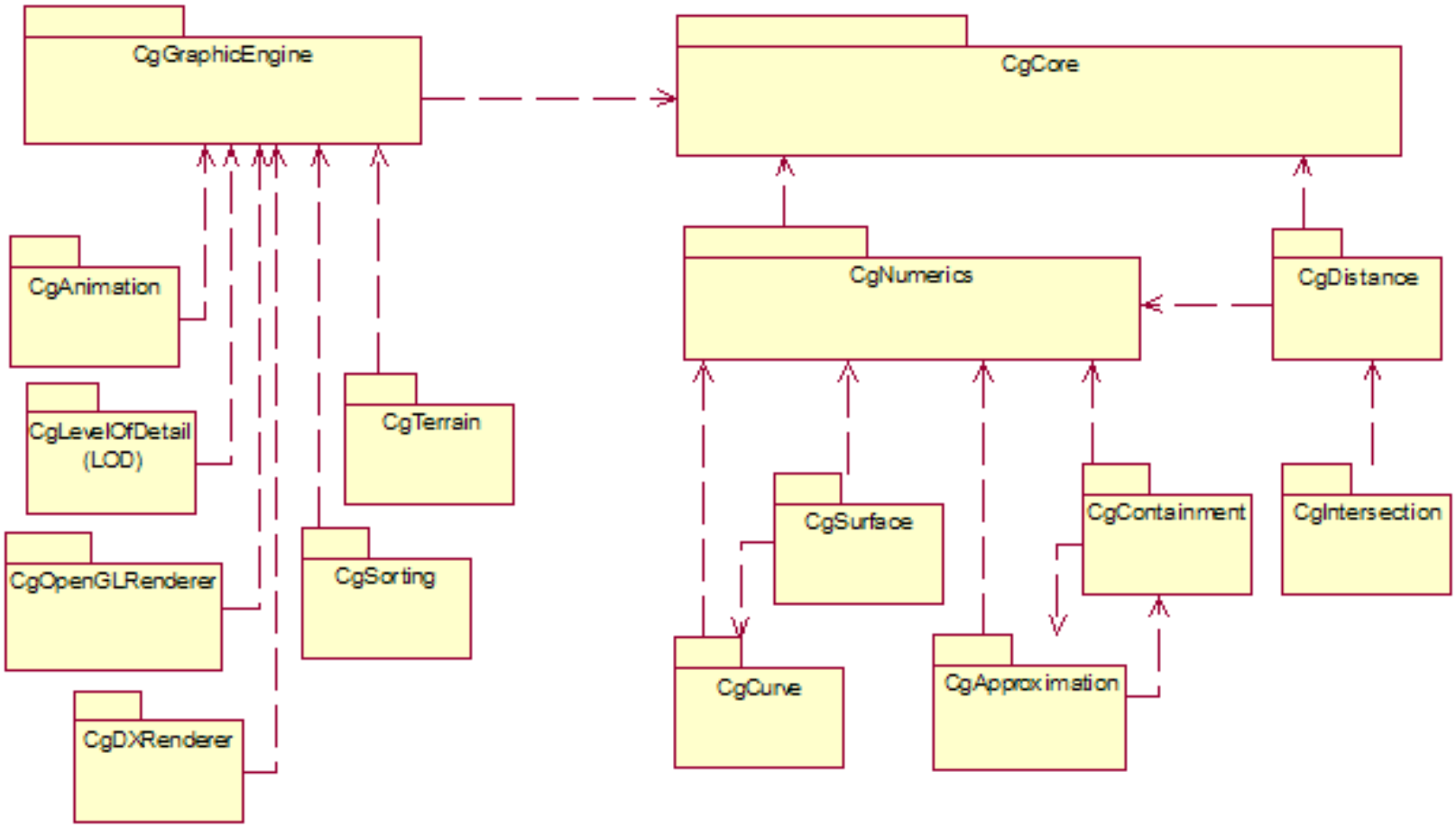
- Un package è un elemento di raggruppamento
- Serve per organizzare elementi UML correlati
- Si applica sia ai diagrammi, sia alle classi e ad altri elementi UML (ad es. i casi d'uso)
- Simbolo:



Esempi di package

- Diagramma di package poco informativo.
- Elenca una serie di package, ma non esplicita relazioni tra i package
- È utile esplicitare le relazioni tra i package, in particolare **dipendenze** e i **livelli d'astrazione**

Esempi di package



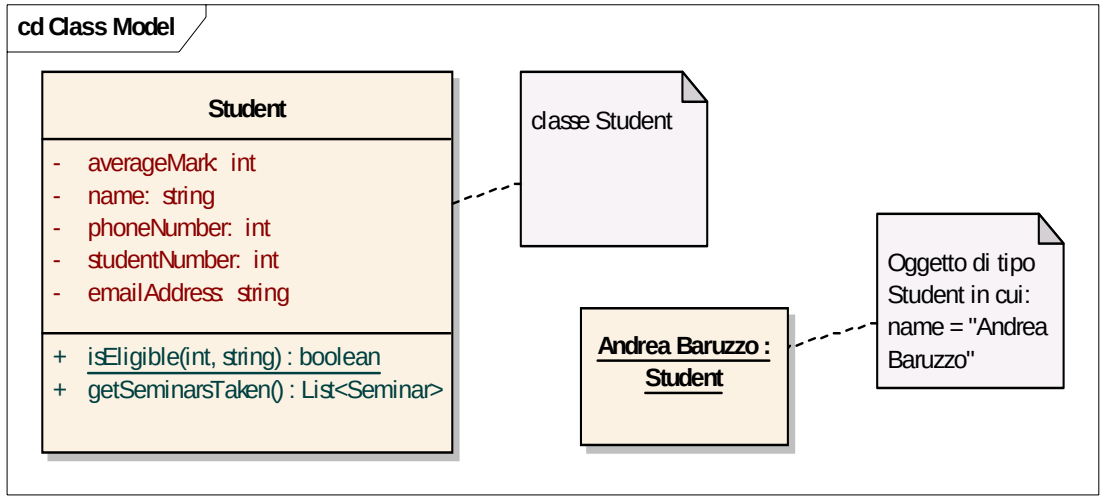
Ispirato da: "**3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics**"
by David H. Eberly - Morgan Kaufmann, September 2000

- Introduzione: approccio e motivazioni
- **Modellare la struttura**
  - Diagrammi di classe
  - Diagrammi degli oggetti
  - Diagrammi dei componenti
  - Diagrammi di deployment
  - Diagrammi delle strutture composte (composite structure) - (UML 2.0)
- Modellare la dinamica

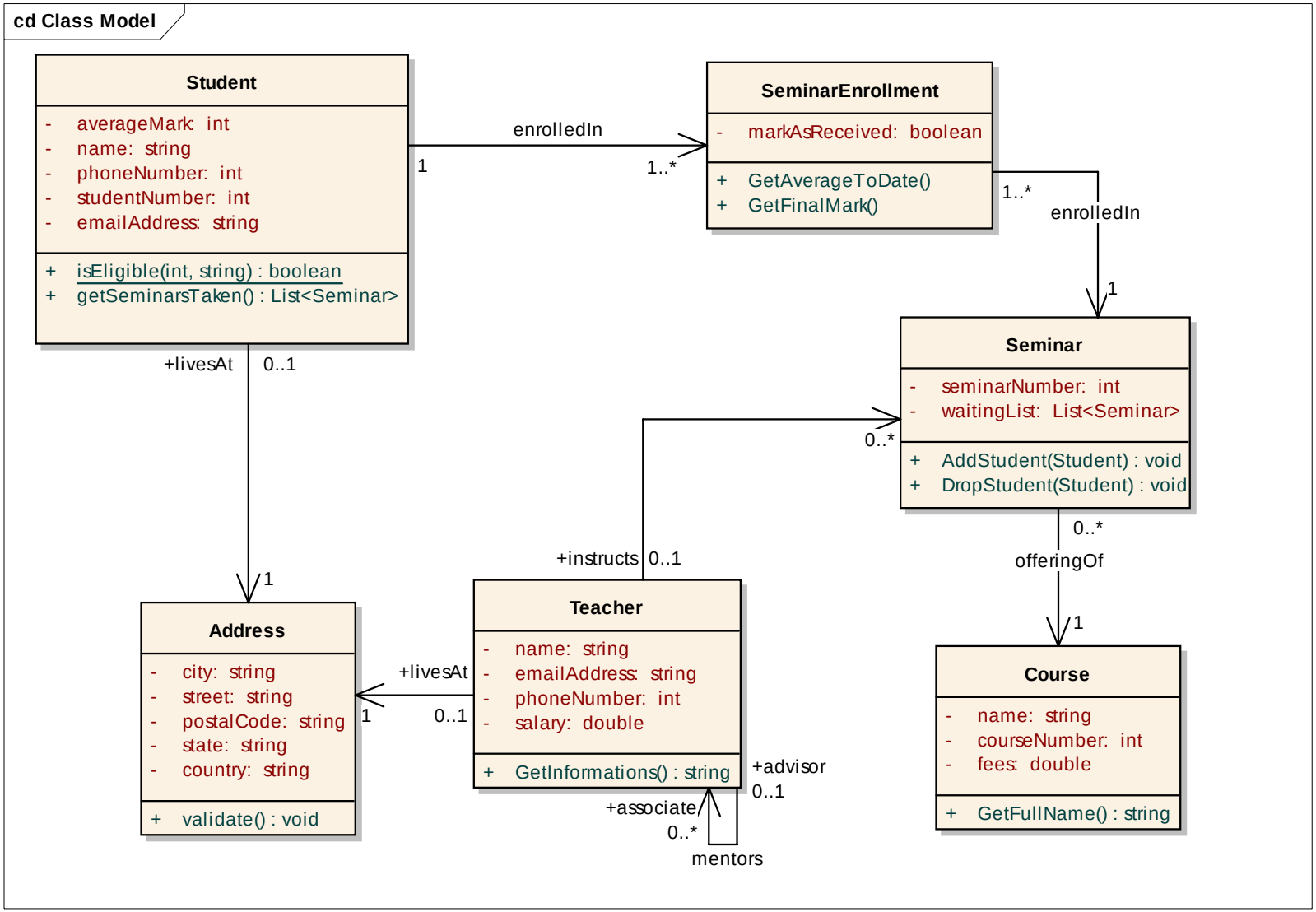
Diagrammi degli oggetti

- E' un *istanza specifica* di un diagramma di classe
- Modella *fatti* ed *esempi*: parla di oggetti specifici
- Le cardinalità nel diagramma di classe vengono sostituite da relazioni esplicite
- Il nome della classe viene preceduto dal nome dell'istanza (oggetto):

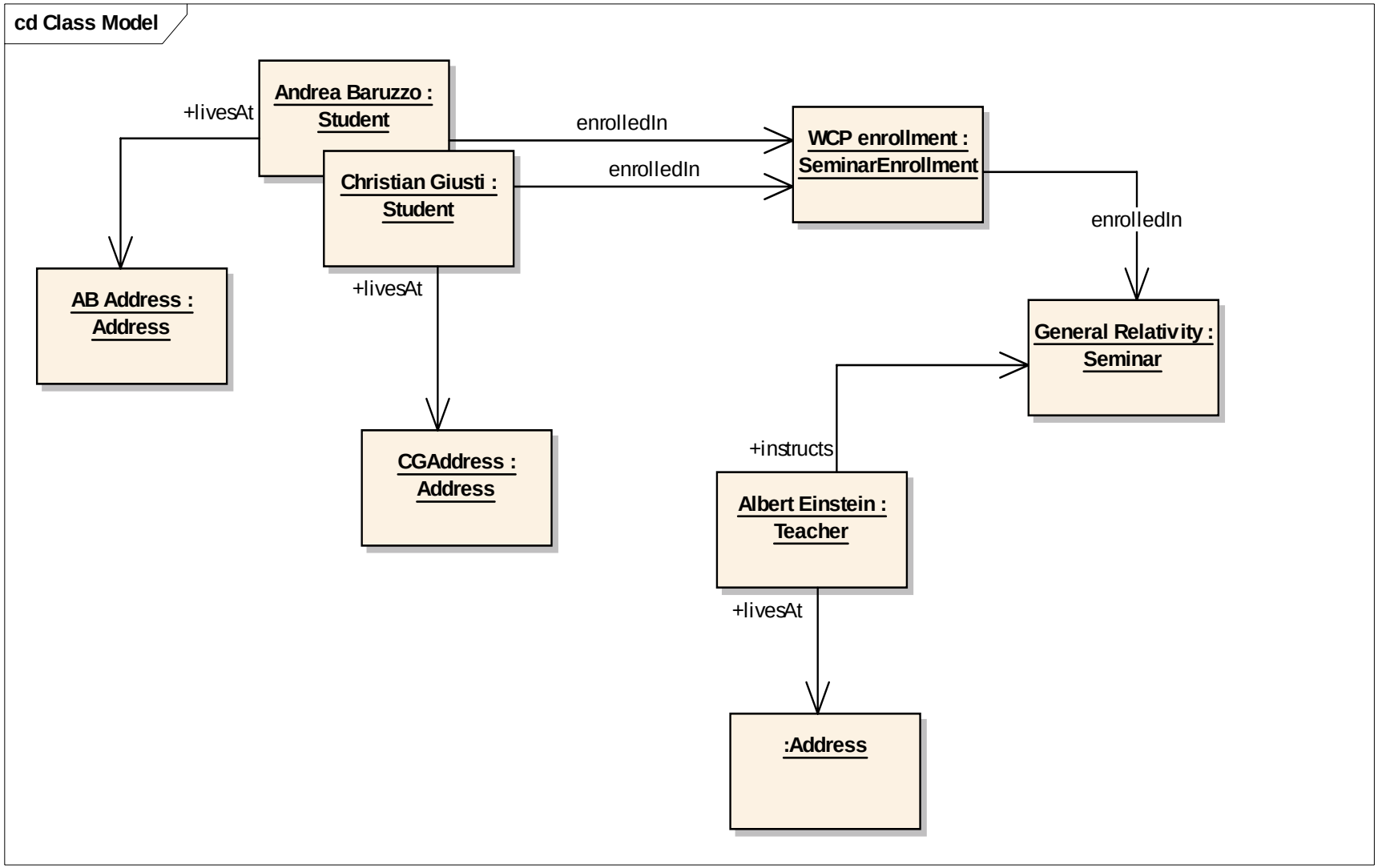
  nomeOggetto : nomeClasse



Istanziare un diagramma di classe mediante un….



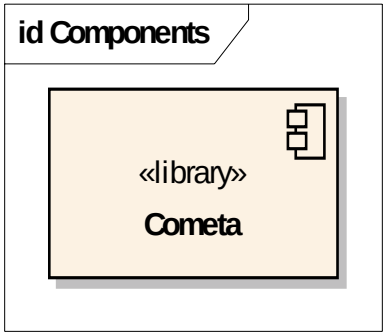
… Diagramma degli oggetti

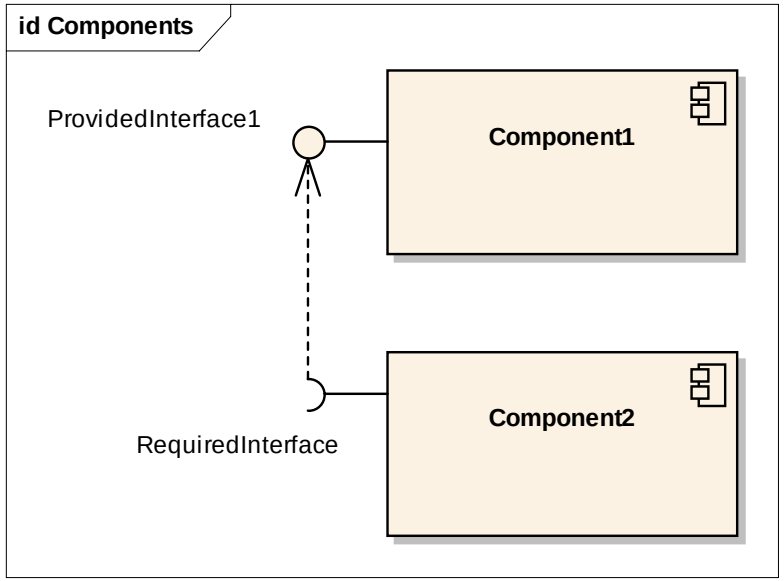


- Introduzione: approccio e motivazioni
- **Modellare la struttura**
  - Diagrammi di classe
  - Diagrammi degli oggetti
  - Diagrammi dei componenti
  - Diagrammi di deployment
  - Diagrammi delle strutture composte (composite structure) – (UML 2.0)
- Modellare la dinamica

Diagrammi dei componenti

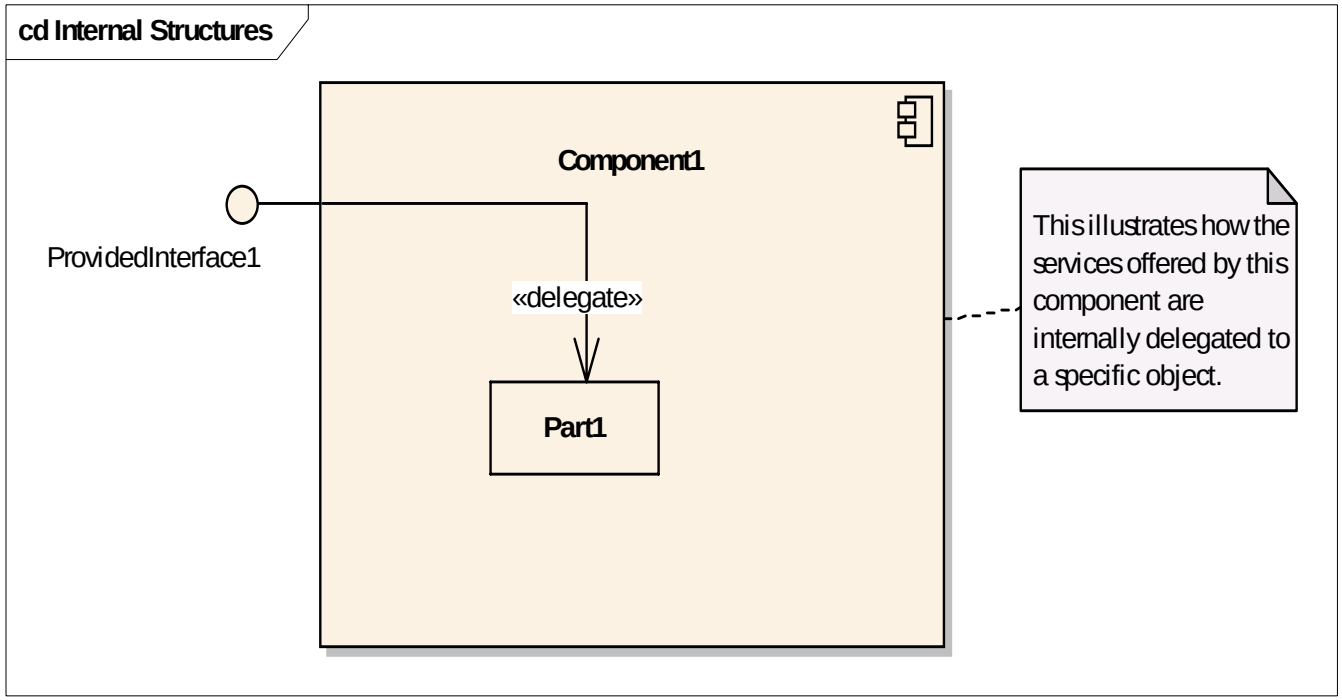
- Descrive le parti fisiche del software (componenti) che costituiscono il sistema nella sua implementazione
- Esempi tipici di componenti: librerie, componenti EJB, form HTML, JSP, browser Web,…
- Evidenzia anche le *relazioni* tra i componenti e la loro *interfaccia*
- Simbolo del componente:



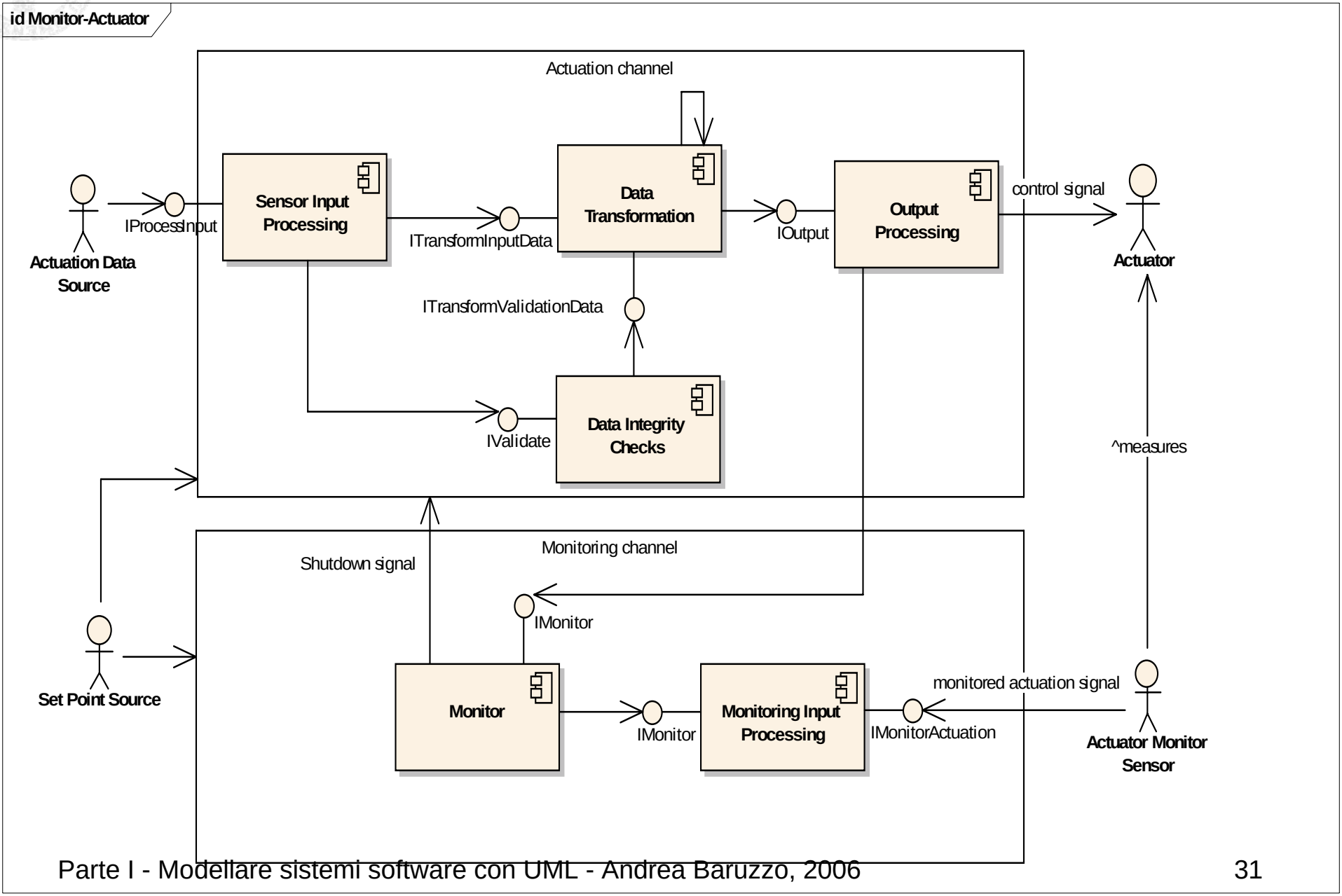


Esempio di diagramma dei componenti

- Diagramma dei componenti che descrive la struttura interna del componente



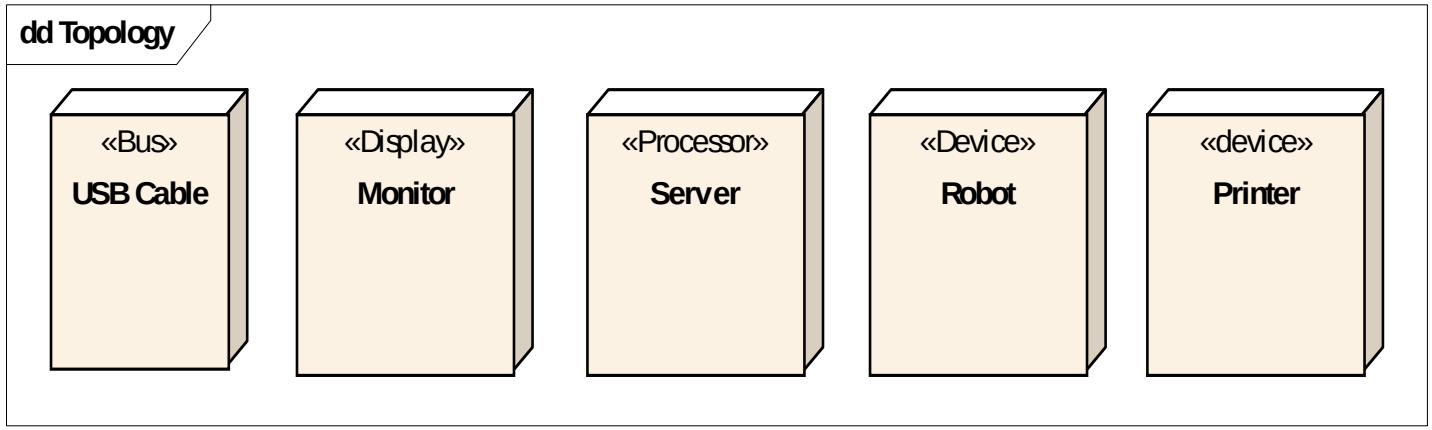
Esempio di diagramma dei componenti



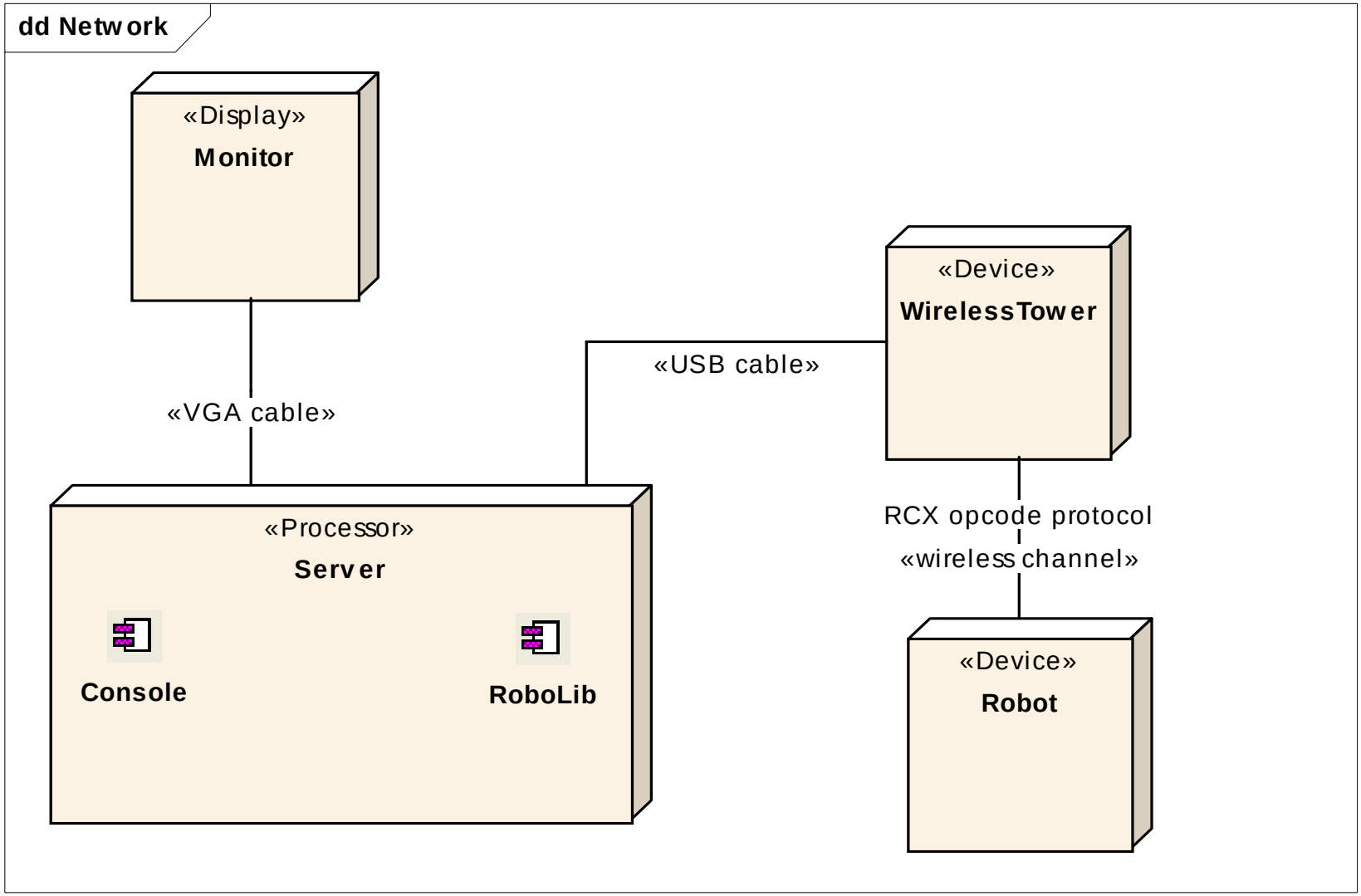
- Introduzione: approccio e motivazioni
- **Modellare la struttura**
  - Diagrammi di classe
  - Diagrammi degli oggetti
  - Diagrammi dei componenti
  - Diagrammi di deployment
  - Diagrammi delle strutture composte (composite structure) – (UML 2.0)
- Modellare la dinamica

Diagrammi di deployment

- Descrive le parti fisiche dell'hardware che costituiscono il sistema nella sua implementazione
- I diversi tipi di hardware (dispositivi) vengono rappresentati mediante nodi
- Le relazioni tra i dispositivi vengono rappresentate mediante associazioni dette connessioni
- Esempi di dispositivi:



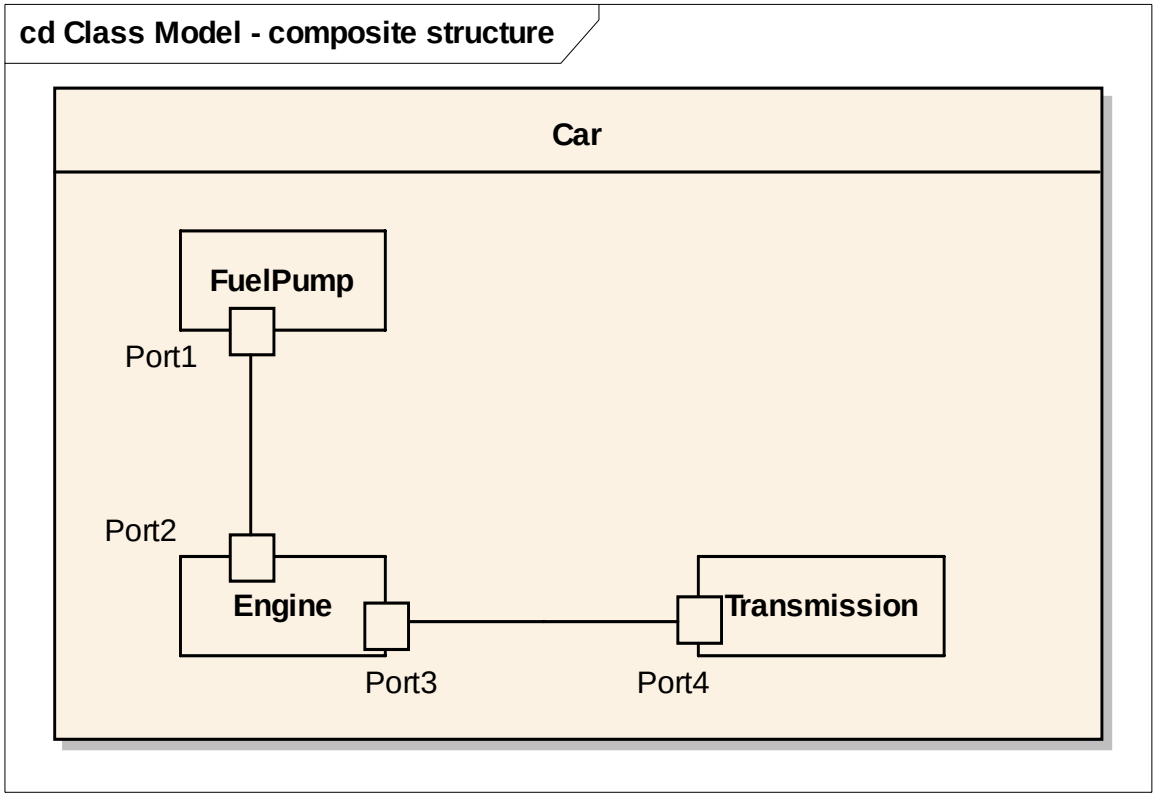
Esempio di diagramma di deployment



- Introduzione: approccio e motivazioni
- **Modellare la struttura**
  - Diagrammi di classe
  - Diagrammi degli oggetti
  - Diagrammi dei componenti
  - Diagrammi di deployment
  - Diagrammi delle strutture composite (composite structure) – (UML 2.0)
- Modellare la dinamica

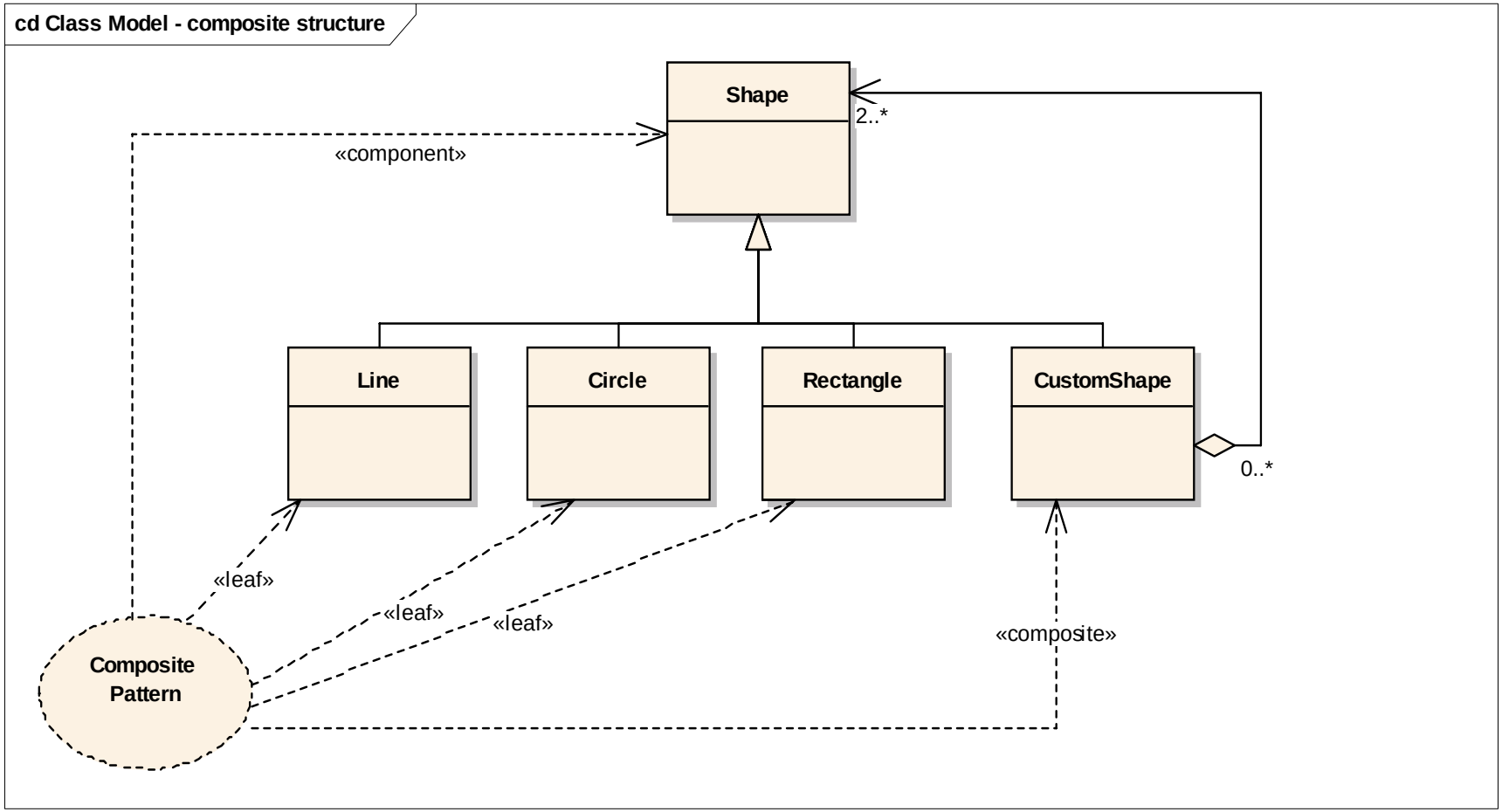
Diagrammi delle strutture composite

- Descrive le classi, le sue parti (aggregati) e il modo in cui tali elementi sono collegati
- Le parti sono collegate mediante connettori e porte



Diagrammi delle strutture composite

- Descrive anche le collaborazioni tra più elementi
- Utile per evidenziare quali classi collaborano assieme per realizzare una particolare interazione (ad es. implementare un design pattern)



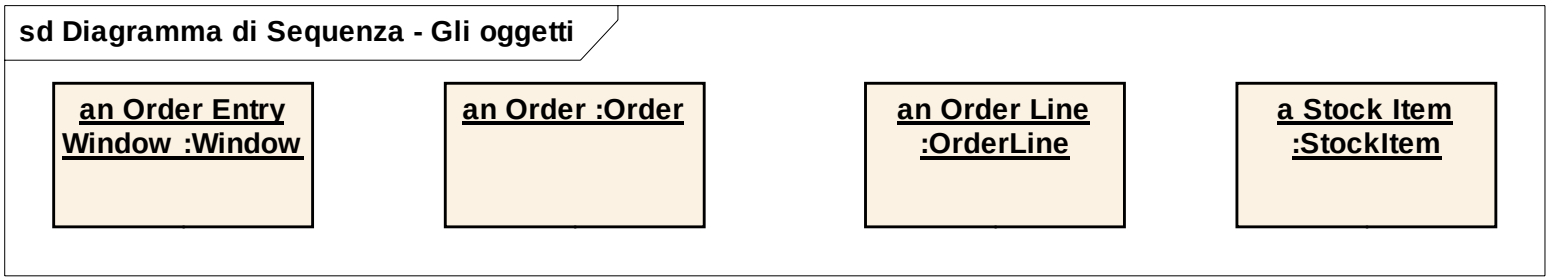
- Introduzione: approccio e motivazioni
- Modellare la struttura
- **Modellare la dinamica**
  - Diagrammi di sequenza
  - Diagrammi di stato
  - Diagrammi di attività
  - Diagrammi dei casi d'uso
  - Diagrammi delle collaborazioni
  - Timing diagram

Diagrammi di sequenza

- I diagrammi di sequenza descrivono le interazioni tra oggetti che collaborano per svolgere un compito
- Sono utili per evidenziare la distribuzione del controllo nel sistema ("chi" fa "che cosa" …)
- Gli oggetti collaborano scambiandosi messaggi
- Lo scambio di un messaggio in OOP equivale all'invocazione di un metodo

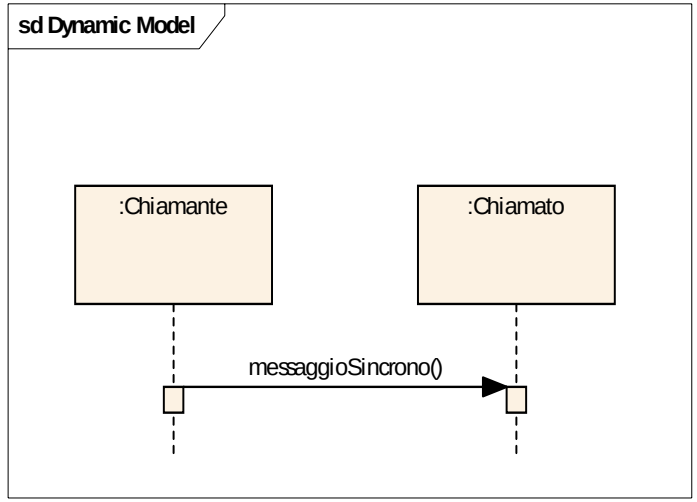
Gli oggetti

- Asse x (asse degli oggetti):
  - Gli oggetti sono disposti orizzontalmente
  - Un oggetto è un'istanza di una classe
  - Sintassi: nomeOggetto : NomeClasse
- Asse t (asse del tempo):
  - Il flusso del tempo è descritto verticalmente



Scambio di messaggi sincroni 1/2

- Si disegna con una freccia chiusa da chiamante a chiamato.
  La freccia è etichettata col nome del metodo invocato e, opzionalmente, con i suoi parametri e il suo valore di ritorno

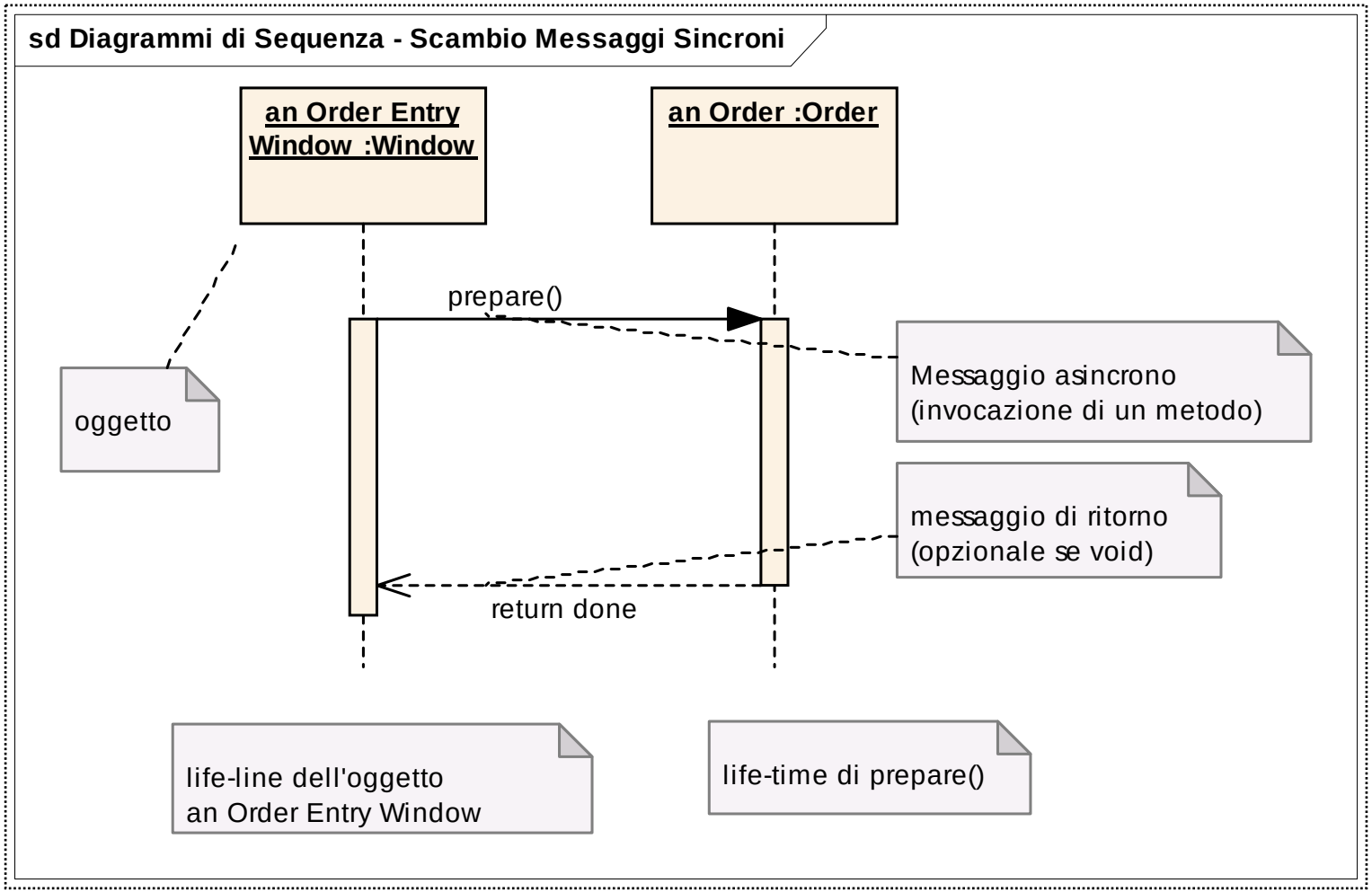


- Il chiamante attende la terminazione del metodo del chiamato prima di proseguire
- Il life-time (durata, vita) di un metodo è rappresentato da un rettangolino che collega la freccia di invocazione con la freccia di ritorno

Scambio di messaggi sincroni 2/2

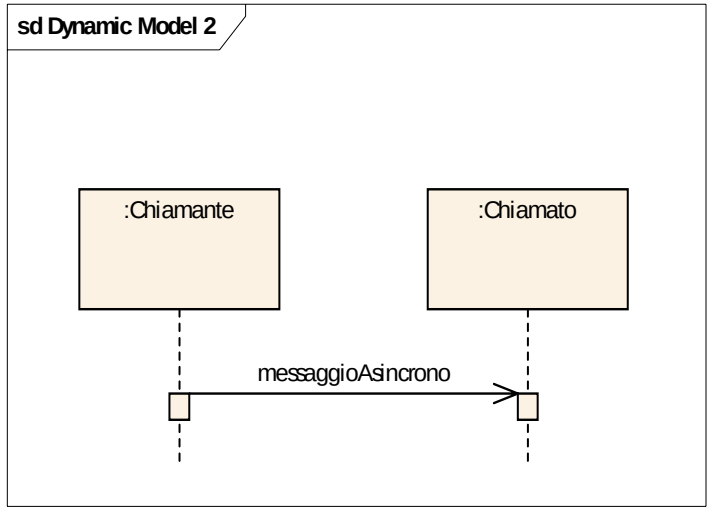
- Life-time corrisponde ad avere un record di attivazione di quel metodo sullo stack di attivazione
- Il ritorno è rappresentato con una freccia tratteggiata
- Il ritorno è sempre opzionale. Se si omette, la fine del metodo è decretata dalla fine del life-time

Scambio di messaggi sincroni - un esempio



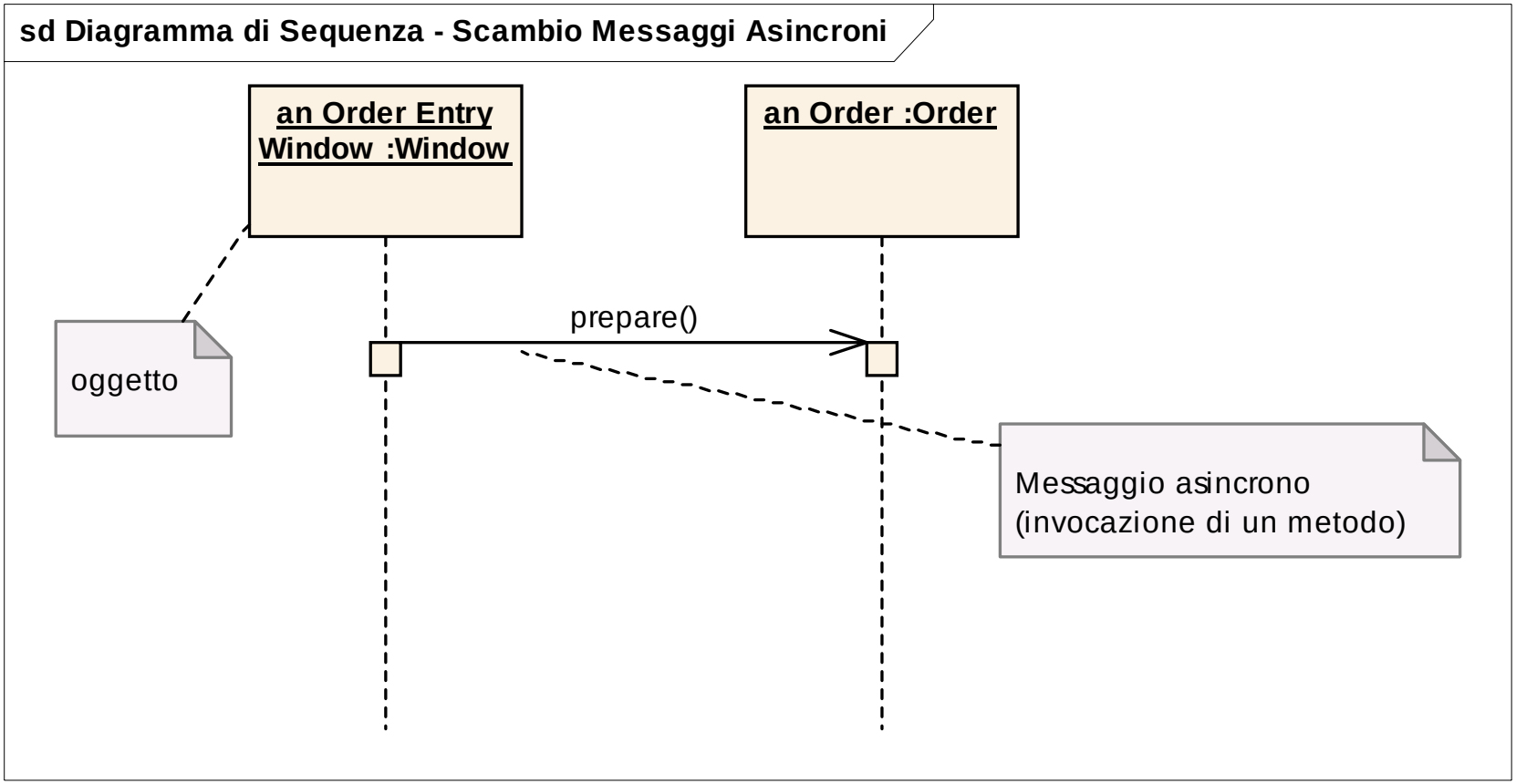
Scambio di messaggi asincroni 1/2

- Si usano per descrivere interazioni concorrenti
- Si disegna con una freccia aperta da chiamante a chiamato.
  La freccia è etichettata col nome del metodo invocato e, opzionalmente, con i suoi parametri e il suo valore di ritorno



- Il chiamante non attende la terminazione del metodo del chiamato, ma prosegue subito dopo l'invocazione
- Il ritorno non segue quasi mai la chiamata

Scambio di messaggi asincroni - un esempio

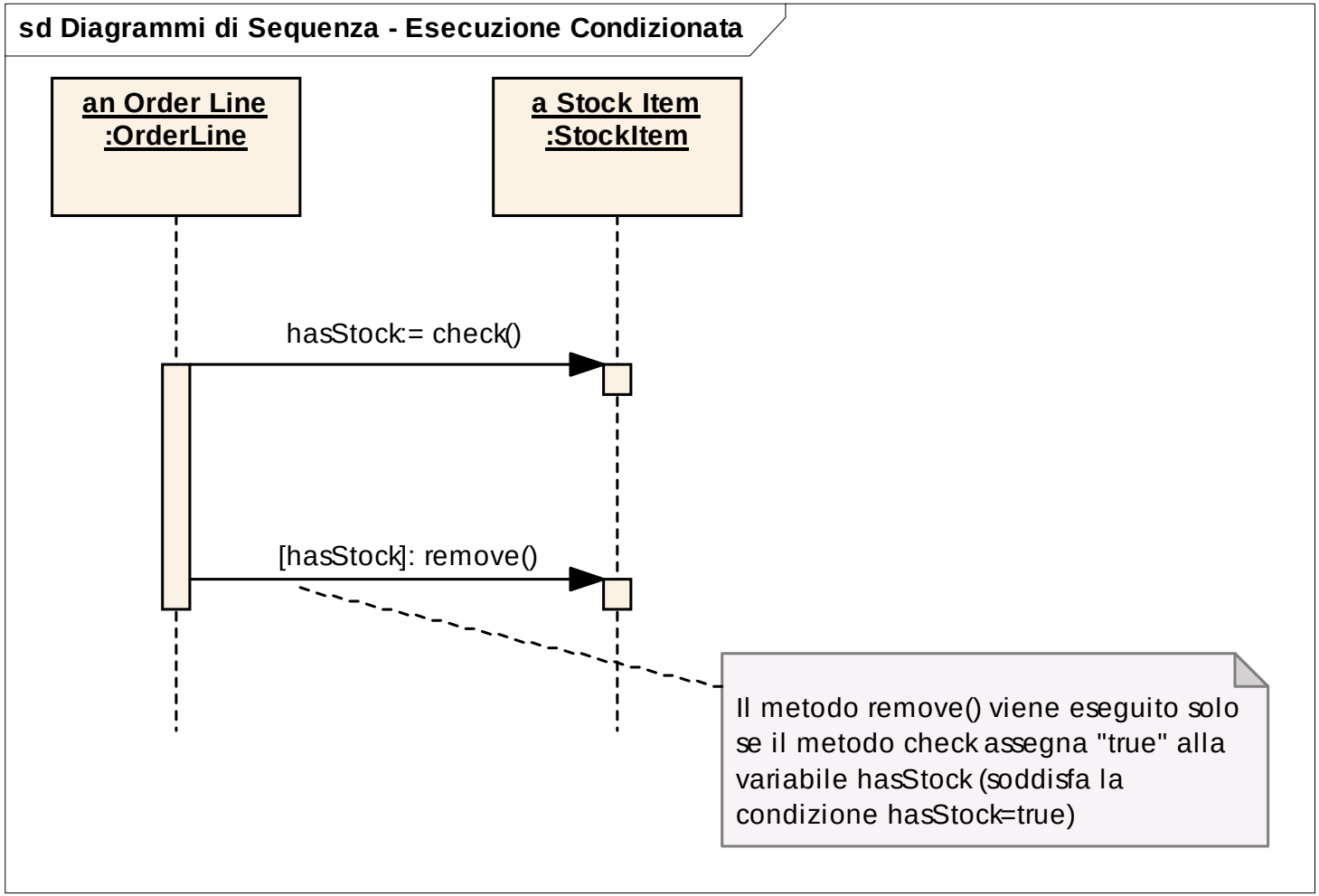


Esecuzione condizionata di un messaggio

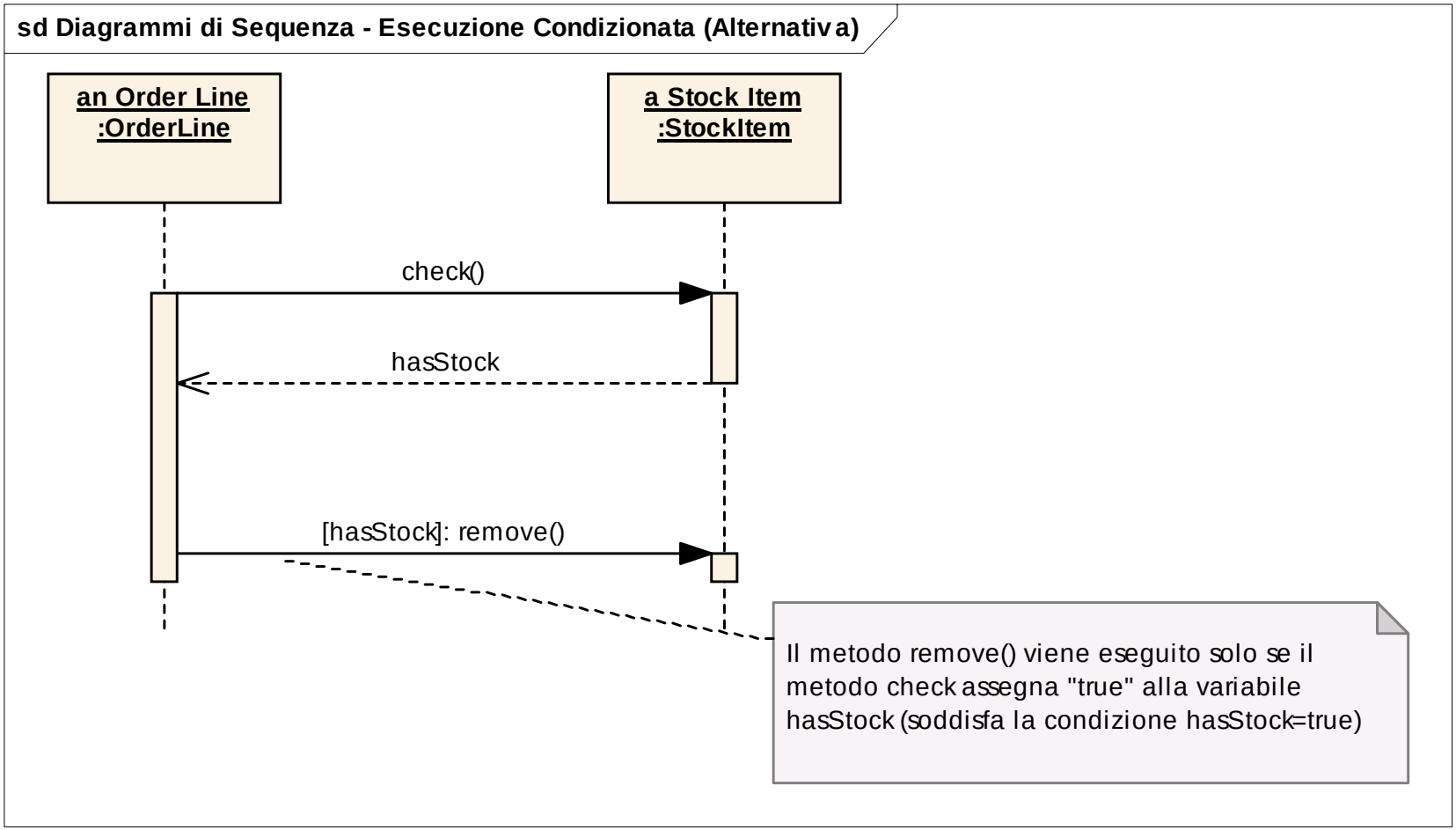
- L'esecuzione di un metodo può essere assoggettata ad una condizione. Il metodo viene invocato solo se la condizione risulta verificata a run-time
- Se la condizione non è verificata, il diagramma *non* dice cosa succede (a meno che non venga esplicitamente modellato ciascun caso)
- La condizione si rappresenta sulla freccia di invocazione del metodo, racchiusa tra parentesi quadre
- Sintassi:

  [cond] : nomeMetodo()

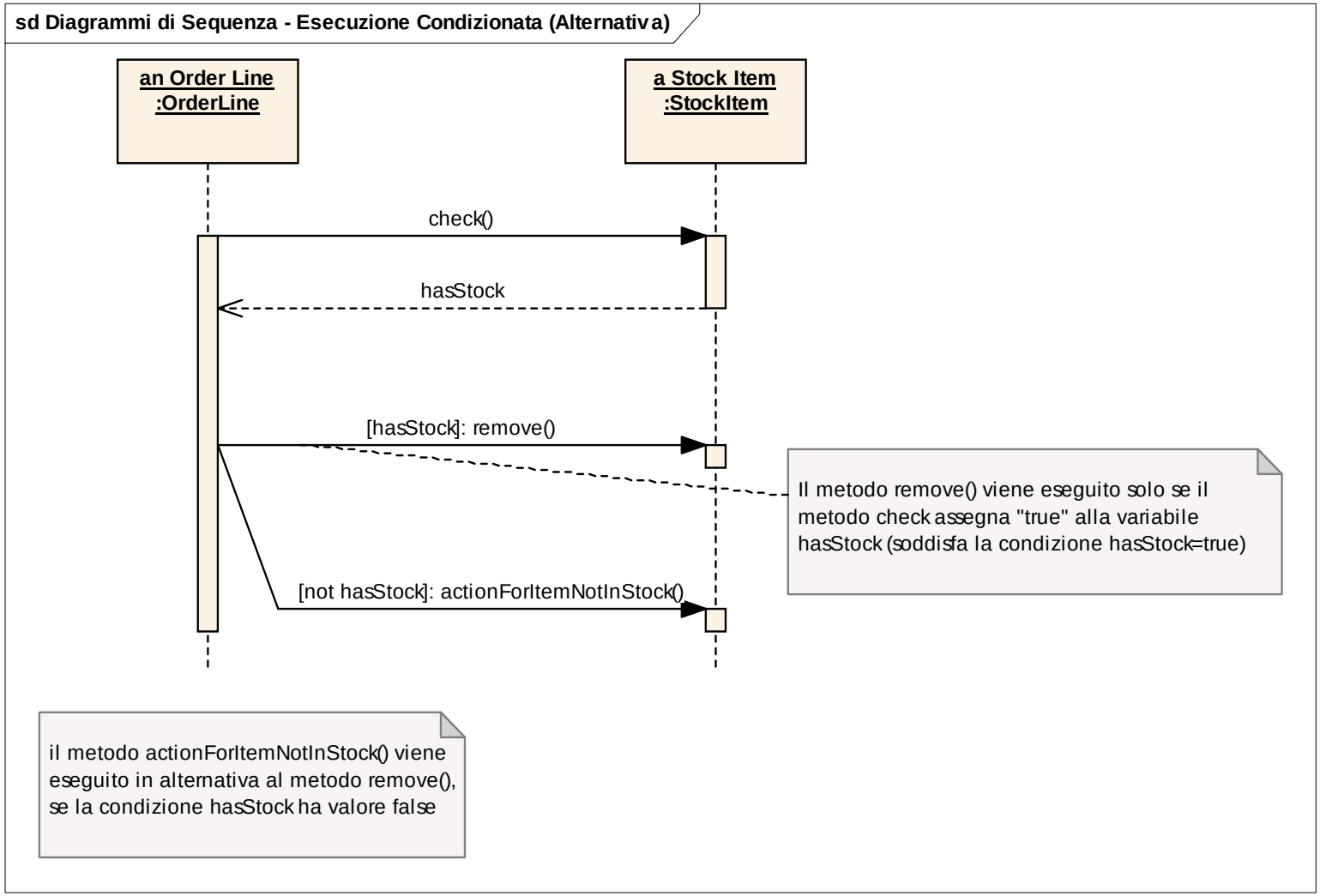
Messaggi condizionati – un esempio



Messaggi condizionati – un esempio (alternativa)



Messaggi condizionati – modellare il caso alternativo

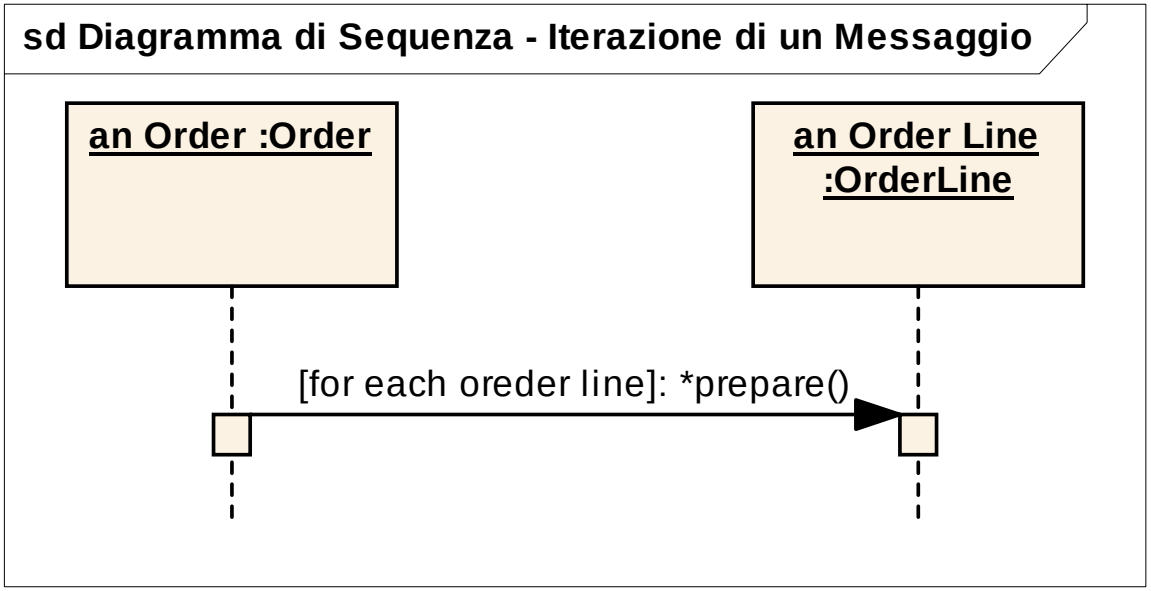


Iterazione di un messaggio

- Rappresenta l'esecuzione ciclica di messaggi
- Si disegna aggiungendo un * (asterisco) prima del metodo su cui si vuole iterare
- Si può aggiungere la condizione che definisce l'iterazione, combinando messaggi condizionati e cicli
- Sintassi (completa di condizione di iterazione):

  [cond] : * nomeMetodo()

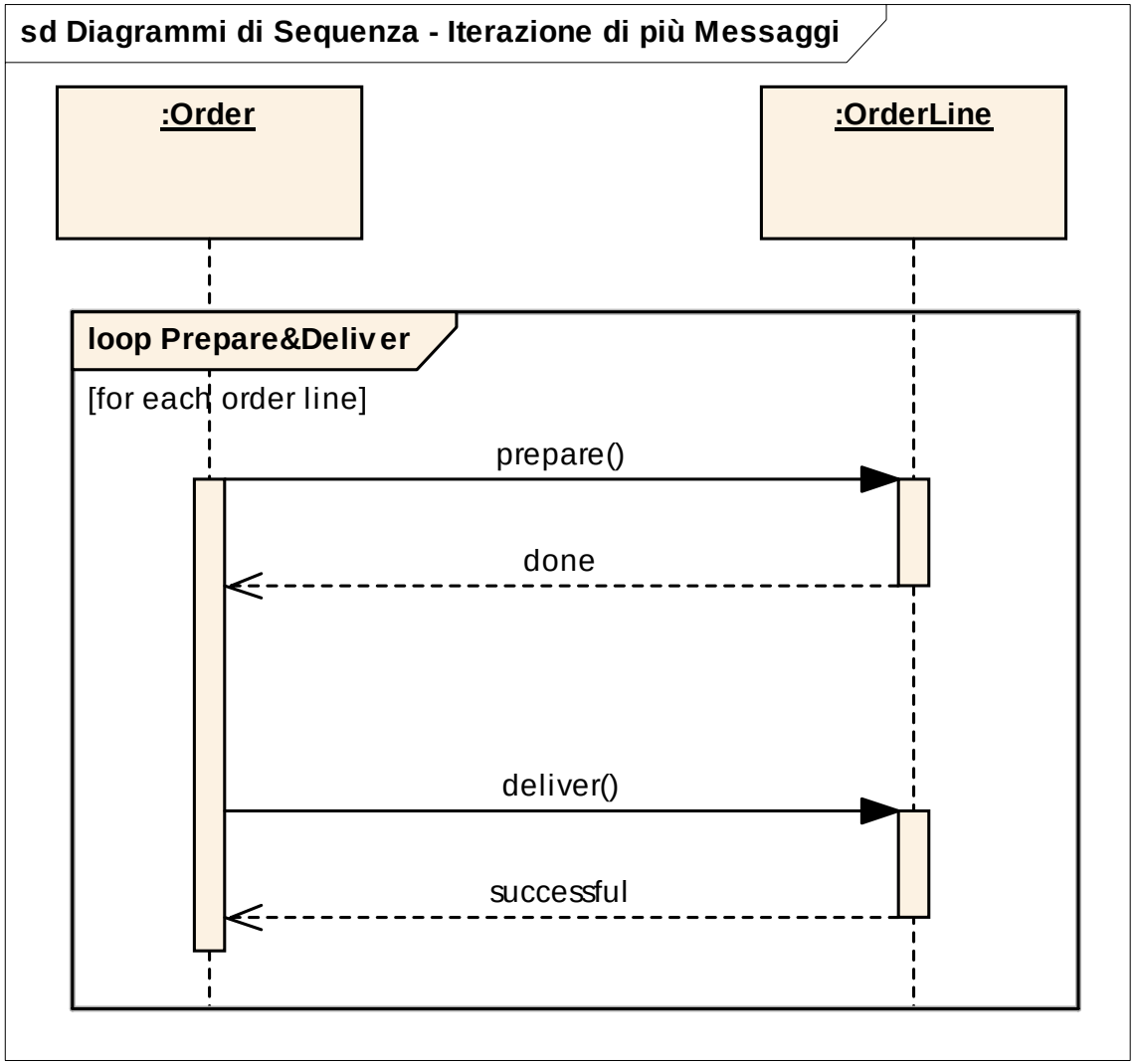
Iterazione di un messaggio – un esempio



Iterazione di un blocco di messaggi

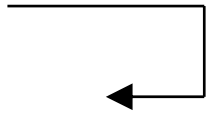
- Rappresenta l'esecuzione ciclica di più messaggi
- Si disegna raggruppando con un blocco (riquadro, box) i messaggi (metodi) su cui si vuole iterare
- Si può aggiungere la condizione che definisce l'iterazione sull'angolo in alto a sinistra del blocco
- La condizione si rappresenta al solito tra parentesi quadre

Iterazione di un blocco di messaggi – un esempio

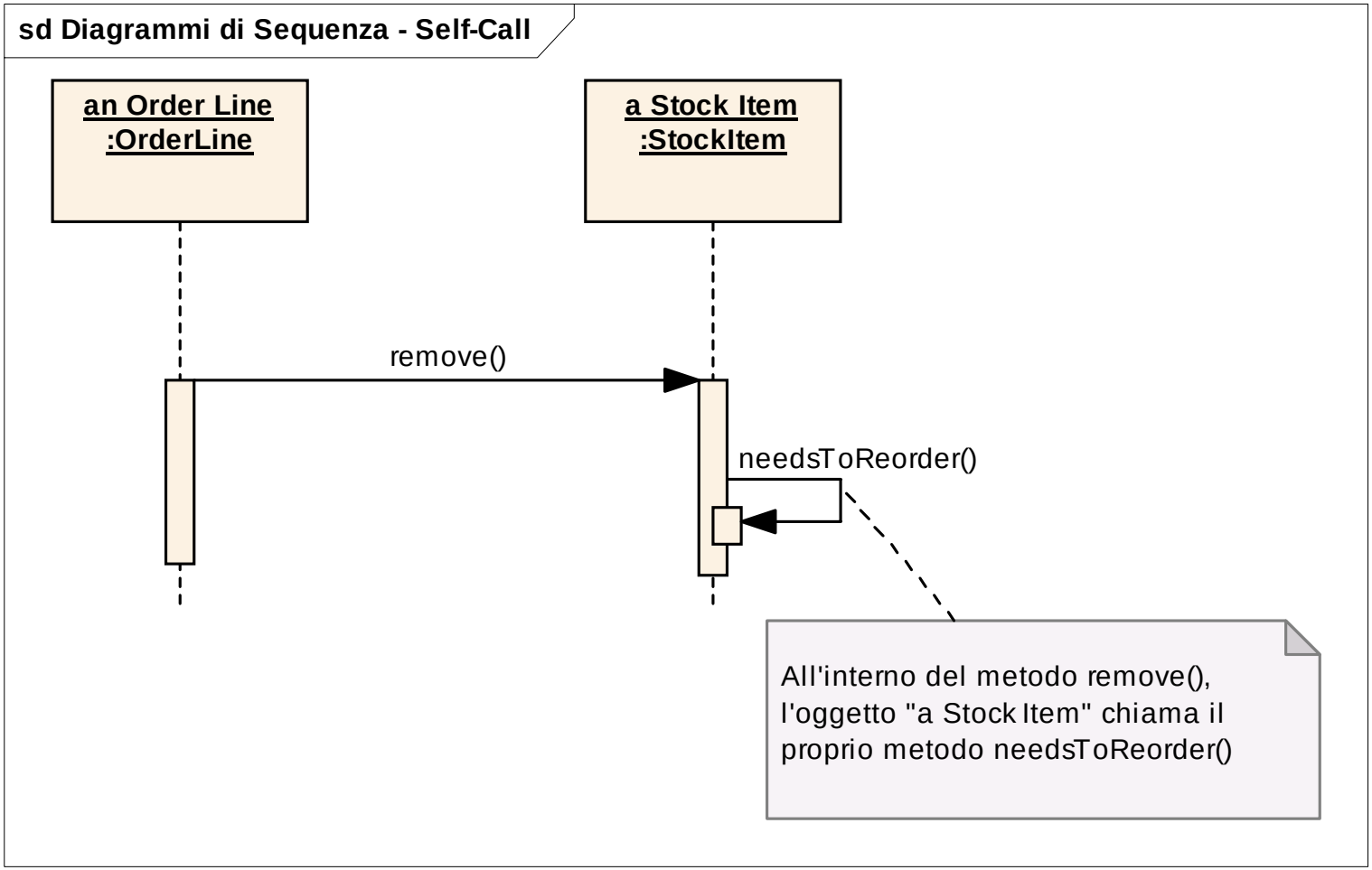


“Auto-chiamata” (self-call)

- Descrive un oggetto che invoca un proprio metodo
- Chiamante e chiamato in questo caso coincidono
- Si rappresenta con una “freccia circolare” che rimane all’interno del life time di uno stesso metodo
- Viene usata anche per rappresentare la ricorsione



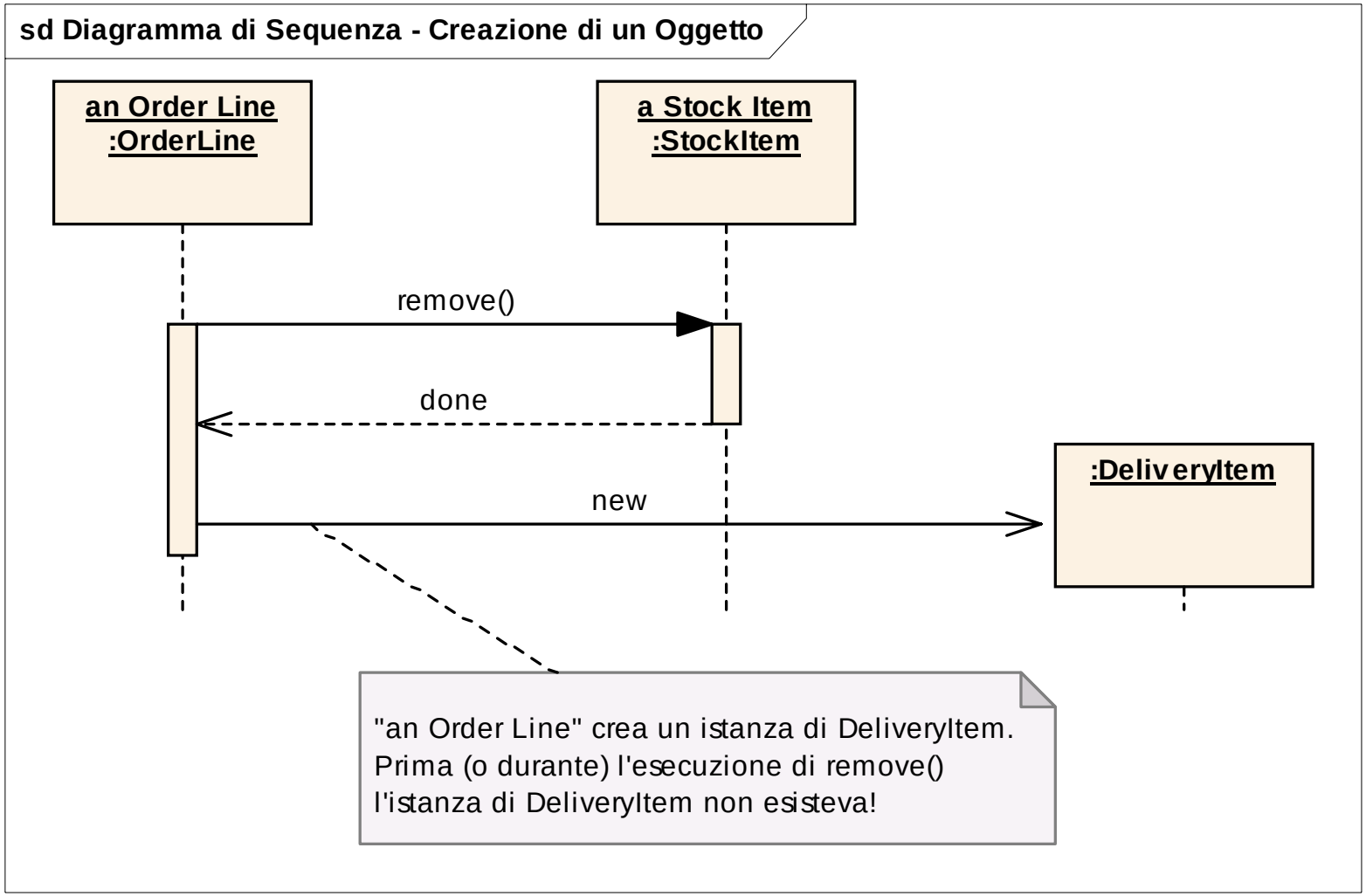
“Auto-chiamata” (self-call) – un esempio



Costruzione di un nuovo oggetto

- Rappresenta la costruzione di un nuovo oggetto non presente nel sistema fino a quel momento
- Corrisponde all'allocazione dinamica (allocazione nello heap di sistema, istruzione new)
- Messaggio etichettato new, create,…
- L'oggetto viene collocato nell'asse temporale in corrispondenza dell'invocazione nel metodo new (o create…)

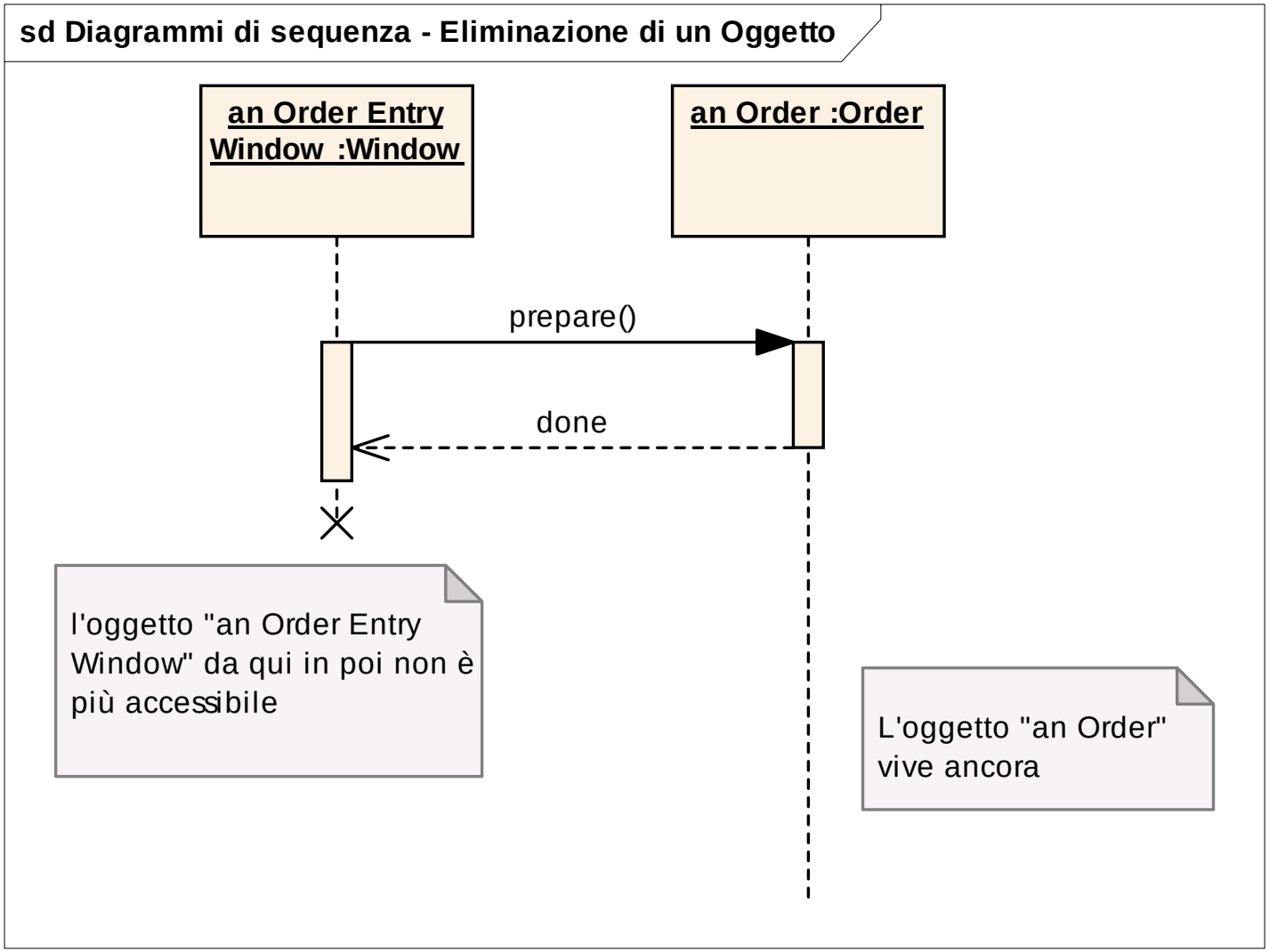
Costruzione di un nuovo oggetto – un esempio



Distruzione di un oggetto (preesistente)

- Rappresenta la distruzione di un oggetto presente nel sistema fino a quel momento
- Corrisponde alla deallocazione dinamica (deallocazione dallo heap di sistema, istruzione delete/dispose/…)
- Si rappresenta con una X posta in corrispondenza della life-line dell'oggetto
- Da quel momento in poi non è "legale" invocare alcun metodo dell'oggetto distrutto

Distruzione di un oggetto preesistente – un esempio



Mettiamo tutto insieme – un esempio completo

- Costruiamo un diagramma di sequenza per il seguente scenario [1]
  - Una finestra di tipo Order Entry invia il messaggio “prepare” ad un Ordine (Order)
  - L’ordine invia il messaggio “prepare” ad ogni sua linea (Order Line)
  - Ogni linea verifica gli elementi in stock (Stock Item)

(continua …)

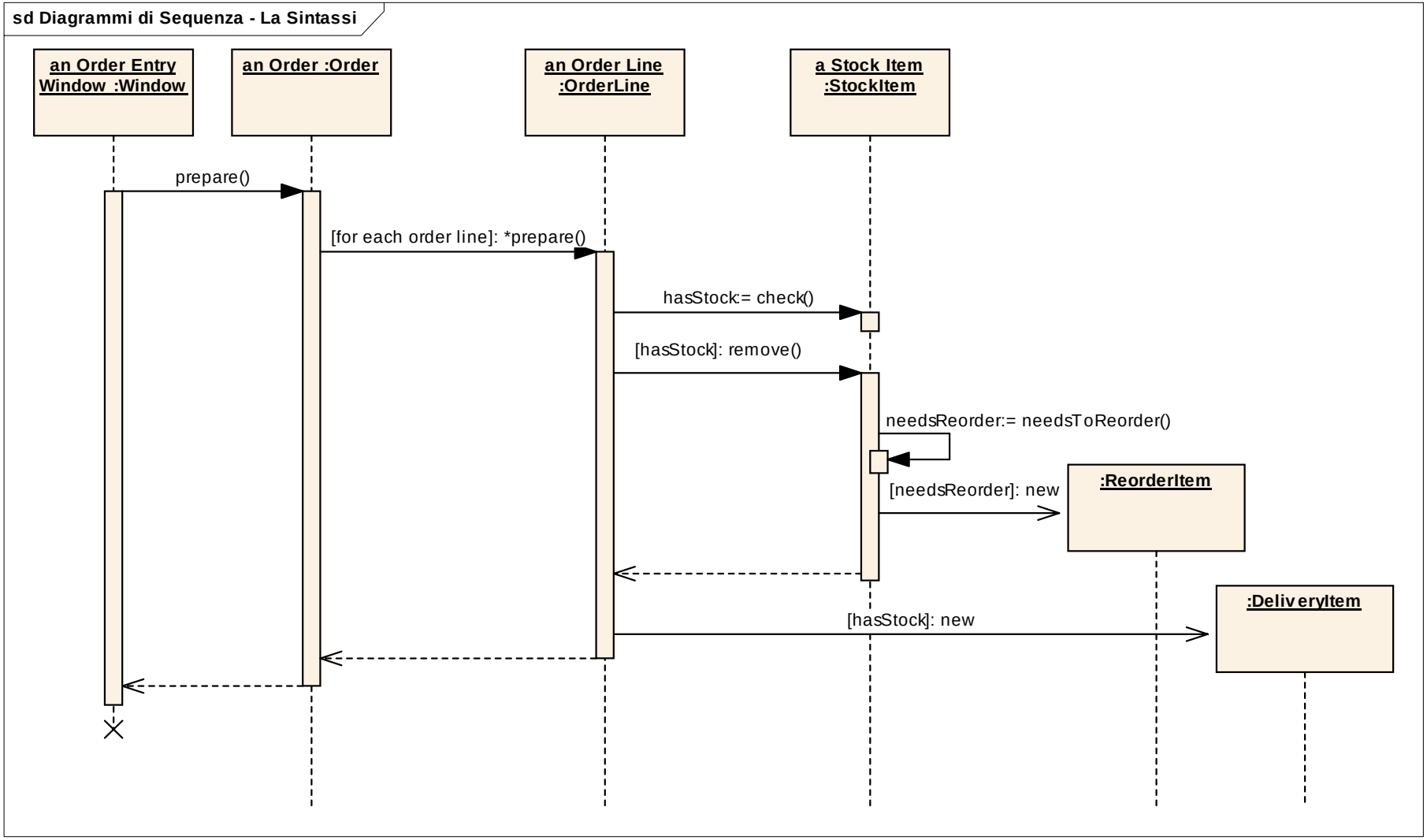
[1] “UML Distilled 3/E”, Martin Fawler, Addson Wesley, 2003

Mettiamo tutto insieme – un esempio completo

(… continua)

- Se il controllo ha esito positivo, la linea rimuove l'appropriata quantità di elementi in stock e crea un'unità di delivery (DeliveryItem)
- Se gli elementi in stock rimanenti scendono al di sotto di una soglia di riordino, viene richiesto un riordino (ReorderItem)

Mettiamo tutto insieme – il diagramma completo



Alcuni suggerimenti finali 1/2

- Assicurarsi che i metodi rappresentati nel diagramma siano gli stessi definiti nelle corrispondenti classi (con lo stesso numero e lo stesso tipo di parametri)
- Documentare ogni assunzione nella dinamica con note o condizioni (ad es. il ritorno di un determinato valore al termine di un metodo, il verificarsi di una condizione all'uscita da un loop, ecc.)
- Mettere un titolo per ogni diagramma (ad es. "sd Diagrammi di Sequenza – Eliminazione di un Oggetto")

Alcuni suggerimenti finali 2/2

- Scegliere nomi espressivi per le condizioni e per i valori di ritorno
- Non inserire troppi dettagli in un unico diagramma (flussi condizionati, condizioni, logica di controllo)
- Non bisogna rappresentare tutto quello che si rappresenta nel codice …
- Se il diagramma è complesso, scomporlo in più diagrammi semplici (ad es. uno per il ramo if, un altro per il ramo else, ecc.)

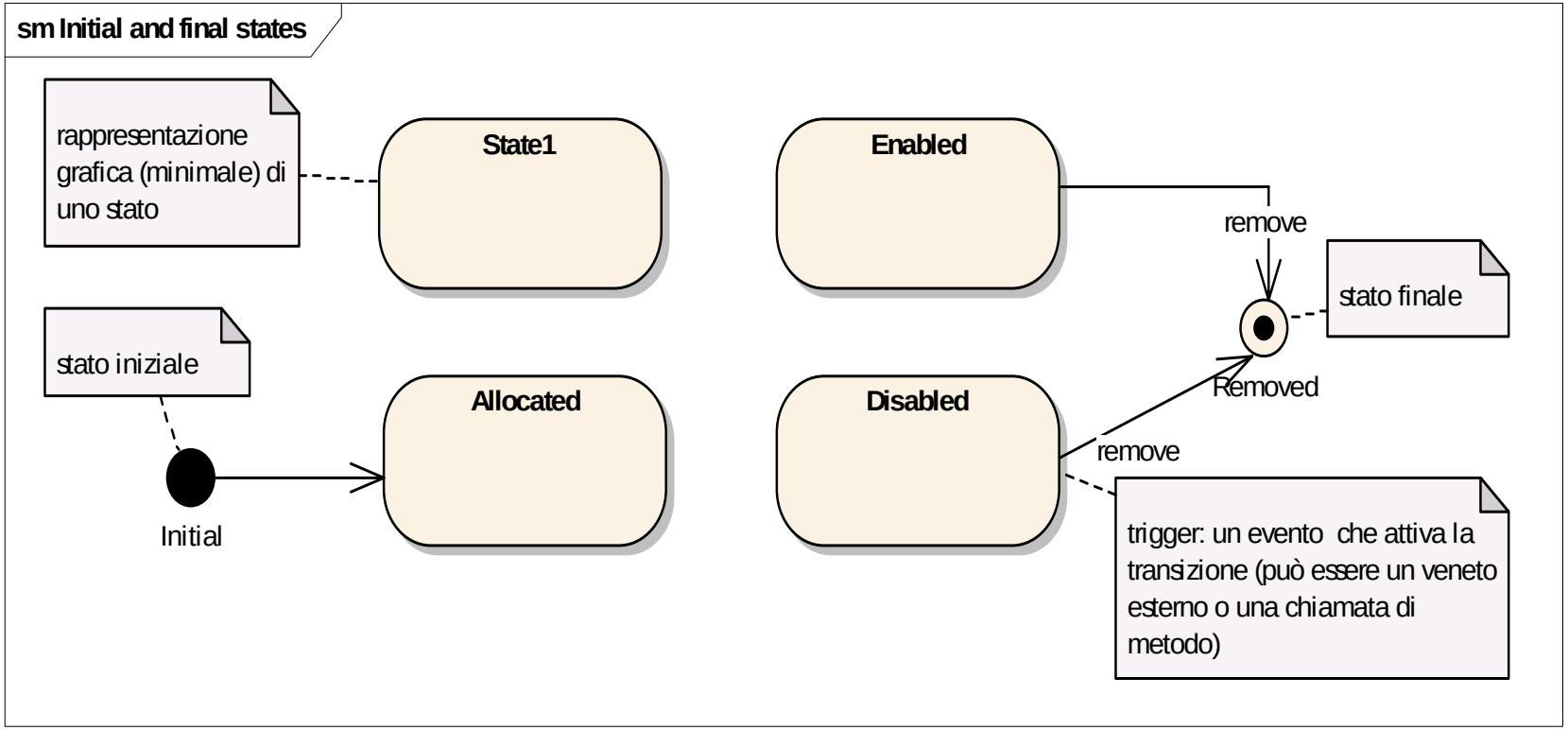
- Introduzione: approccio e motivazioni
- Modellare la struttura
- **Modellare la dinamica**
  - Diagrammi di sequenza
  - Diagrammi di stato (statechart)
  - Diagrammi di attività
  - Diagrammi dei casi d'uso
  - Diagrammi delle collaborazioni
  - Timing diagram

Diagrammi di stato - Definizioni

- Un diagramma di stato rappresenta il ciclo di vita degli oggetti di una classe
- Il ciclo di vita è descritto in termini di
  - Eventi
  - Stati
  - Transizioni di stato
- Gli eventi possono attivare delle transizioni di stato
- Un evento in uno statechart corrisponde ad un messaggio in un sequence diagram
- Uno stato è costituito da un insieme di “valori significativi” assunti dagli attributi dell’oggetto che ne influenzano il comportamento

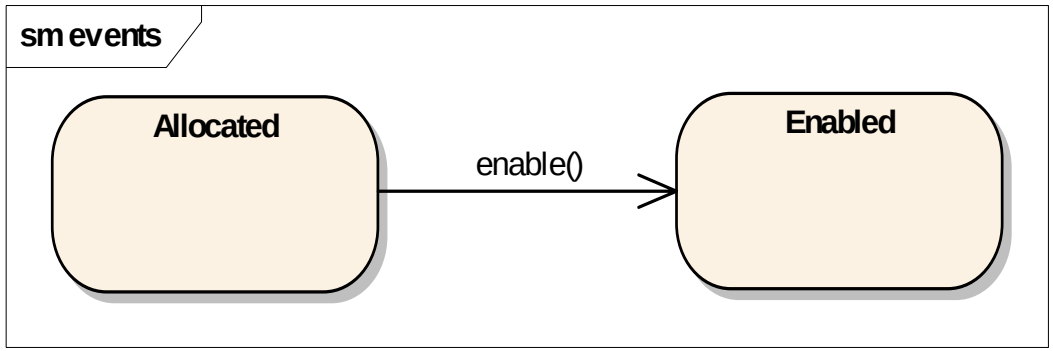
Diagrammi di stato – Stati iniziale e finale

- Esistono due stati “speciali”, detti *pseudostati*:
  - Lo stato iniziale
  - Lo stato finale
- Un oggetto può non avere uno stato finale! (non viene mai distrutto)



Diagrammi di stato – Modellare gli eventi

- Un evento può essere:
  - L'invocazione sincrona di un metodo (una "call")
  - La ricezione di una chiamata asincrona ("signal") – ad esempio la notifica di un'eccezione lanciata
  - Una condizione predefinita che diventa vera (si parla in questo caso di "change event")
  - La fine di un "periodo di tempo" come quello impostato da un timer ("elapsed-time event")
- Un evento si può rappresentare graficamente con una freccia (transizione) etichettata con il nome del metodo o della condizione associata all'evento stesso



Diagrammi di stato – Modellare gli eventi

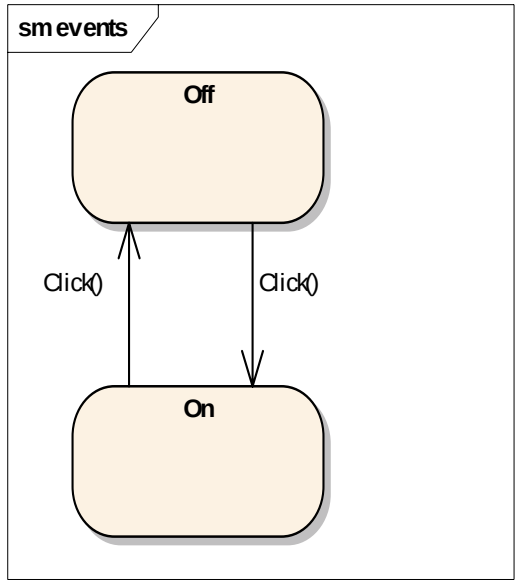
- Un evento può essere rappresentato anche mediante un'espressione testuale avente la seguente sintassi:
  - event-name '(' [comma-separated-parameter-list] ')' ['['guard-condition']'] / [action-expression]

  dove:
  - event-name identifica l'evento
  - parameter-list definisce i valori dei dati che possono essere passati come parametro con l'evento
  - guard-condition determina se l'oggetto che riceve l'evento deve rispondere ad esso (ossia eseguire il metodo associato)
  - action-expression definisce come l'oggetto ricevente deve rispondere all'evento

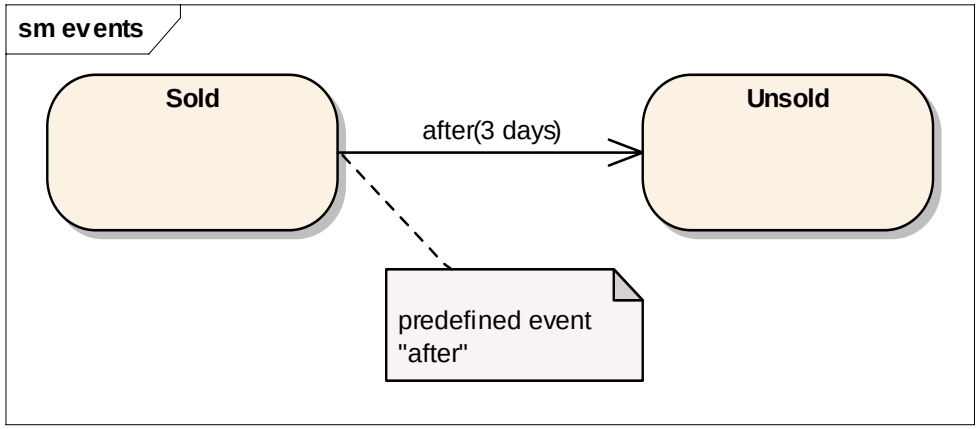
Diagrammi di stato – Esempi

- Event + state = response

  Lo stesso evento causa diversi comportamenti in base allo stato in cui l'oggetto che riceve l'evento si trova

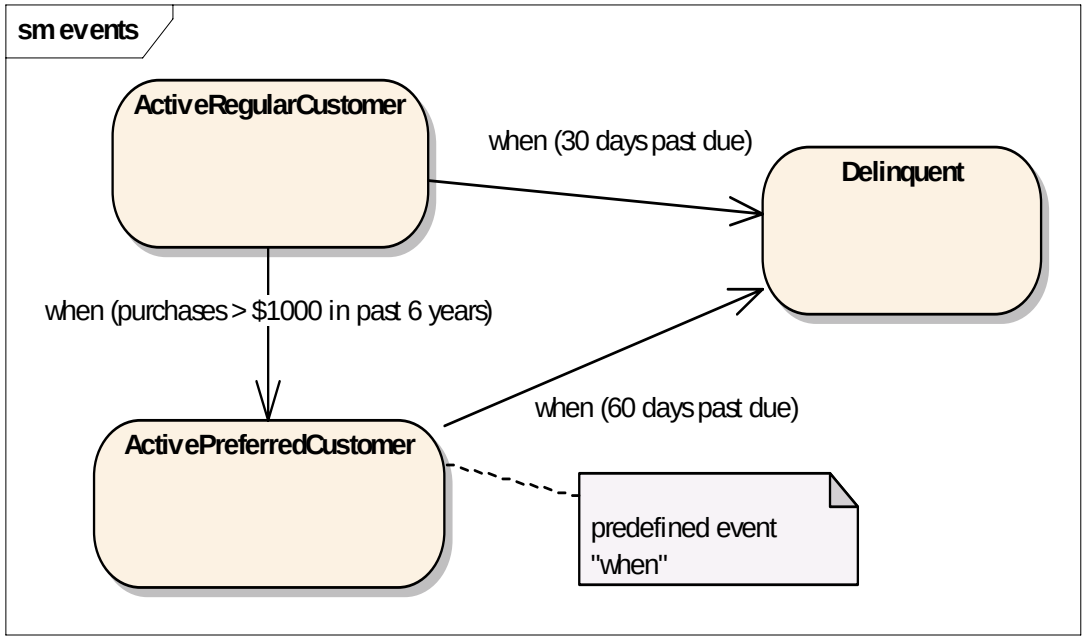


Elapsed-time Events

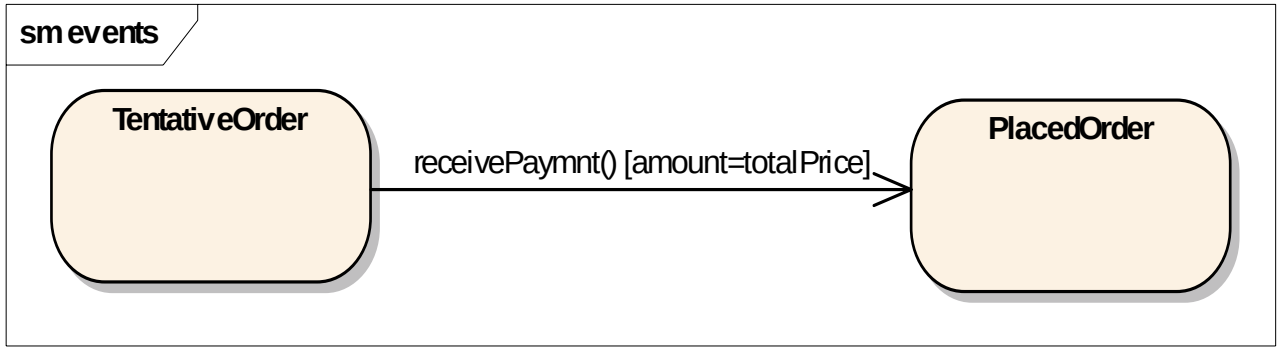


Diagrammi di stato – Esempi

- Change event

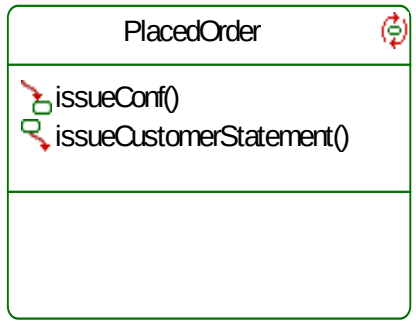


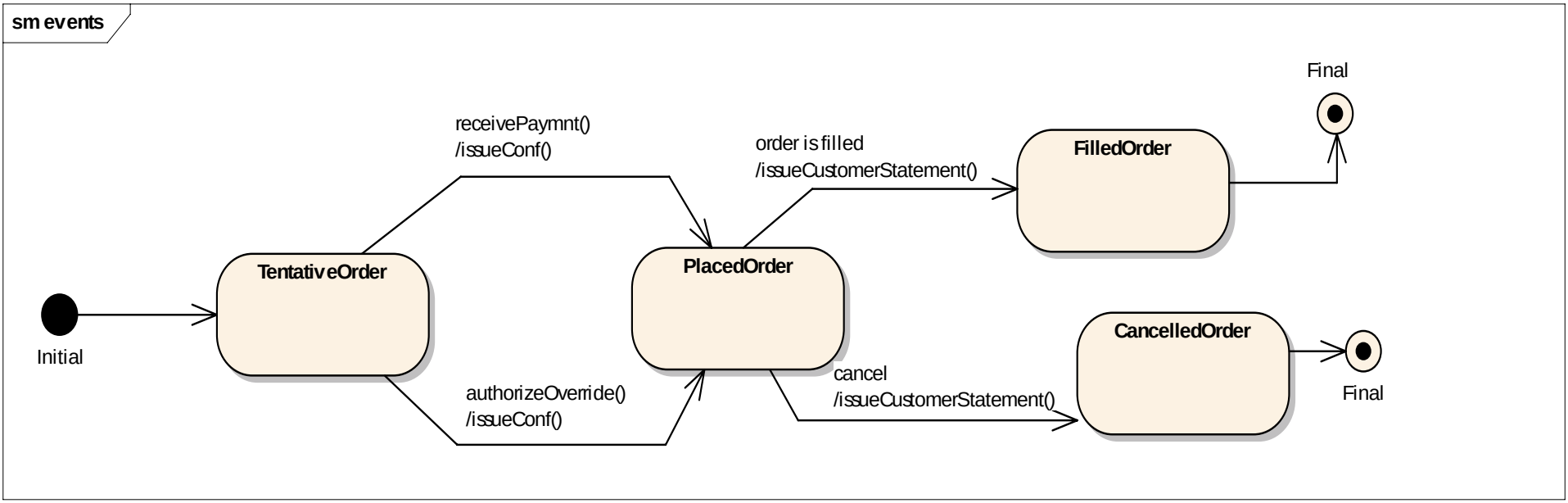
- Guarded event



Diagrammi di stato – Entry/Exit Actions

- Entry/exit action: azione che viene eseguita per ogni evento che causa una transizione entrante/uscente nello/dallo stato
- entry action: azione che viene eseguita in una transizione entrante nello stato
- exit action: azione che viene eseguita in una transizione uscente dallo stato





Diagrammi di stato – Modellare attività

- All'interno degli stati posso essere eseguite delle attività
- Negli statechart distinguiamo tra
  - *Azioni*: operazioni atomiche
  - *Attività*: operazioni generalmente non atomiche
  - Le azioni provocano un cambiamento di stato (entry/exit) e quindi non possono essere interrotte
  - Le attività non alterano lo stato dell'oggetto

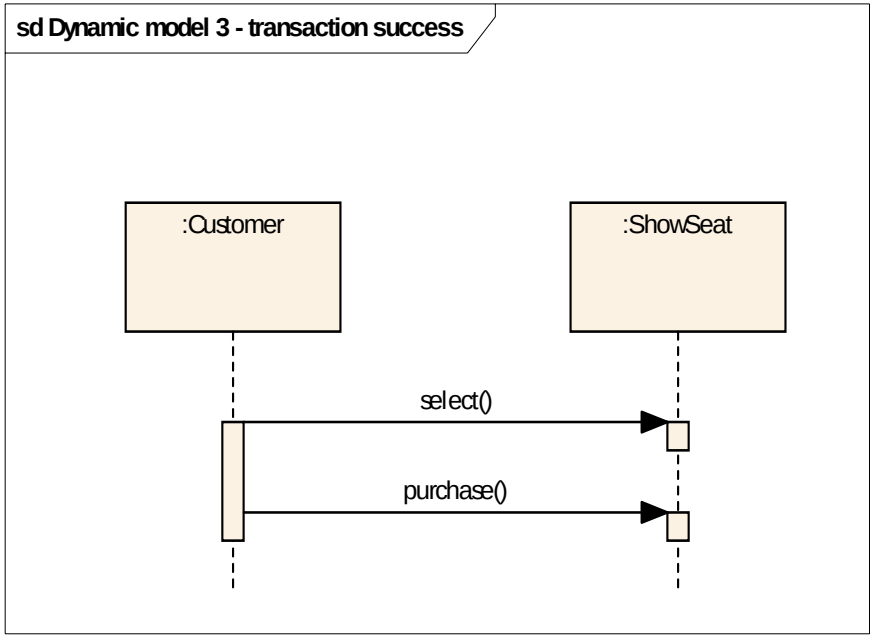
Diagrammi di stato – Ordine di esecuzione degli eventi

- Quando si verifica un evento associato ad una transizione, l'ordine di esecuzione è il seguente:
  1. Se è in esecuzione un'attività, questa viene interrotta ("gracefully" se possibile)
  2. Si esegue l'exit action
  3. Si esegue l'azione associata all'evento
  4. Si esegue l'entry action del nuovo stato
  5. Si inizia l'esecuzione delle eventuali attività del nuovo stato

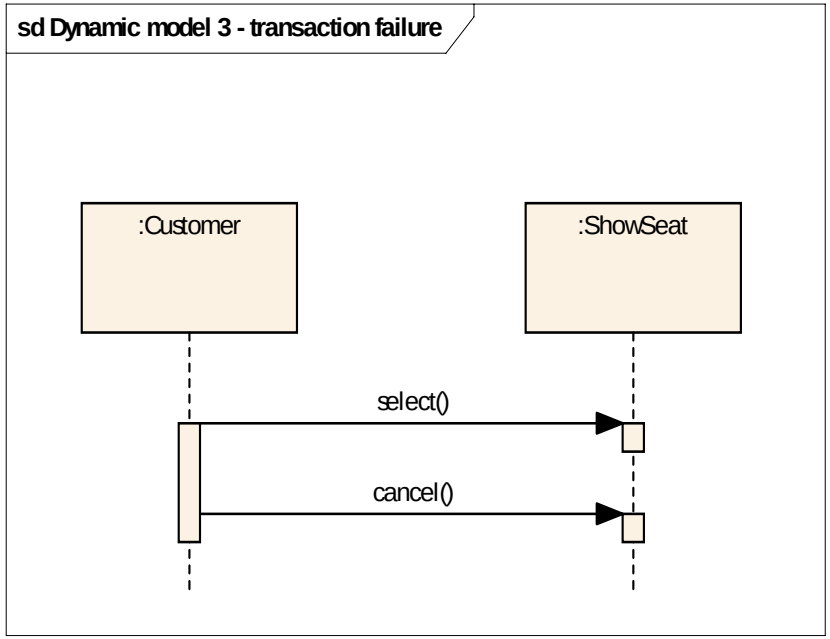
Relazione tra diagrammi di stato e diagrammi di sequenza 1/4

- Due scenari (sequence diagram): successo e fallimento di una transazione

(Successo)



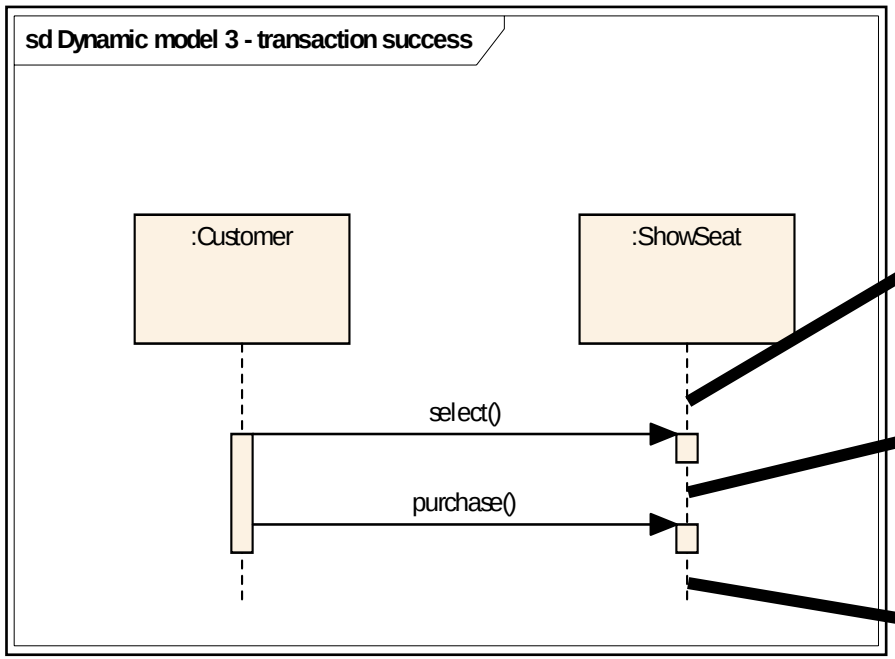
(Fallimento)



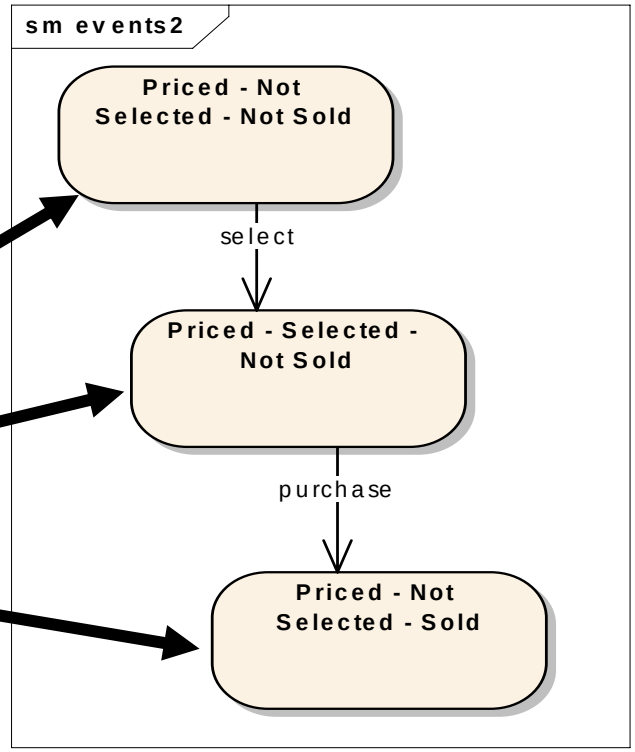
Relazione tra diagrammi di stato e diagrammi di sequenza 2/4

- Scenario di successo e relativo (parziale) diagramma a stati

(Successo)



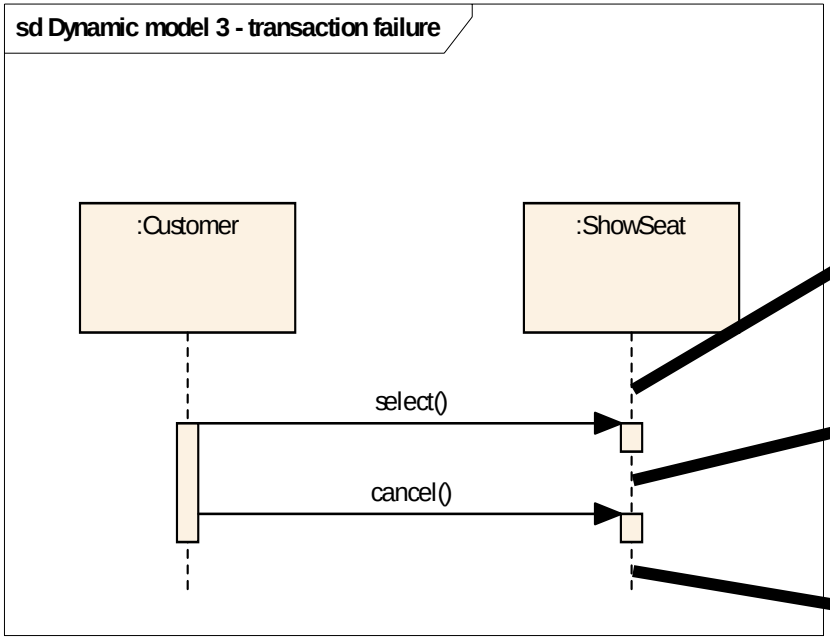
(diagramma a stati parziale)



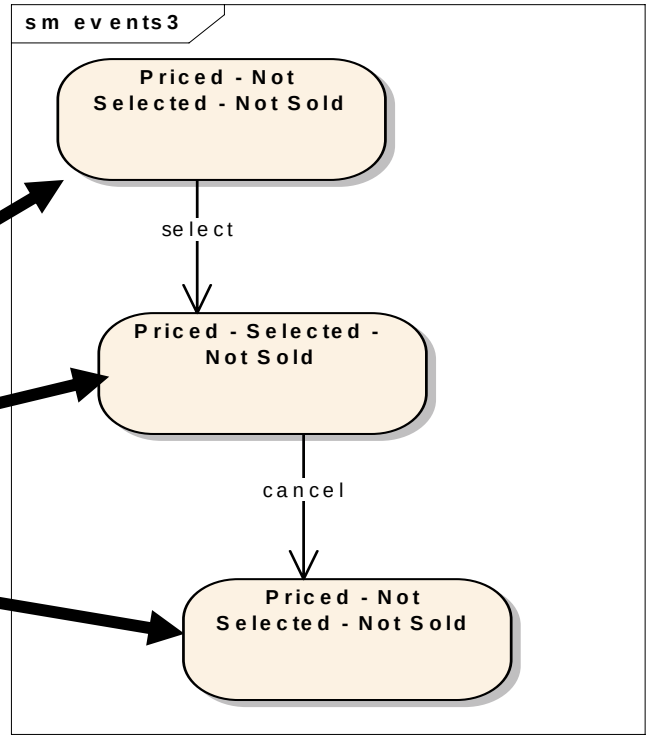
Relazione tra diagrammi di stato e diagrammi di sequenza 3/4

- Scenario di fallimento e relativo (parziale) diagramma a stati

(Fallimento)

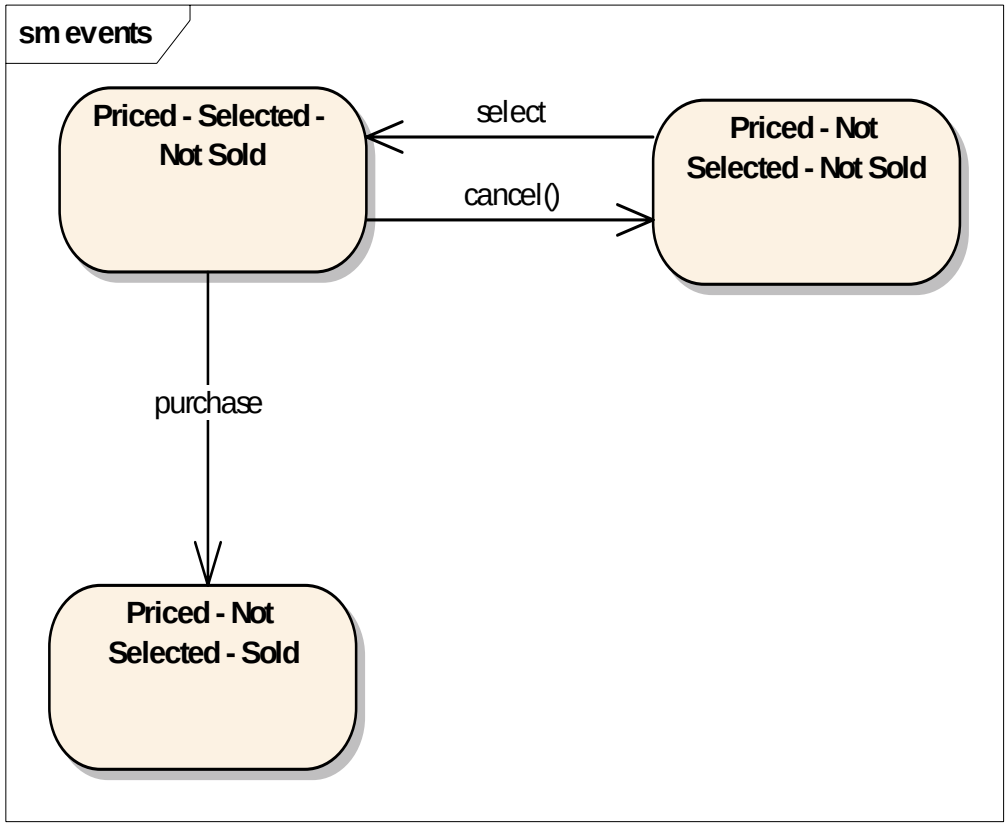


(diagramma a stati parziale)



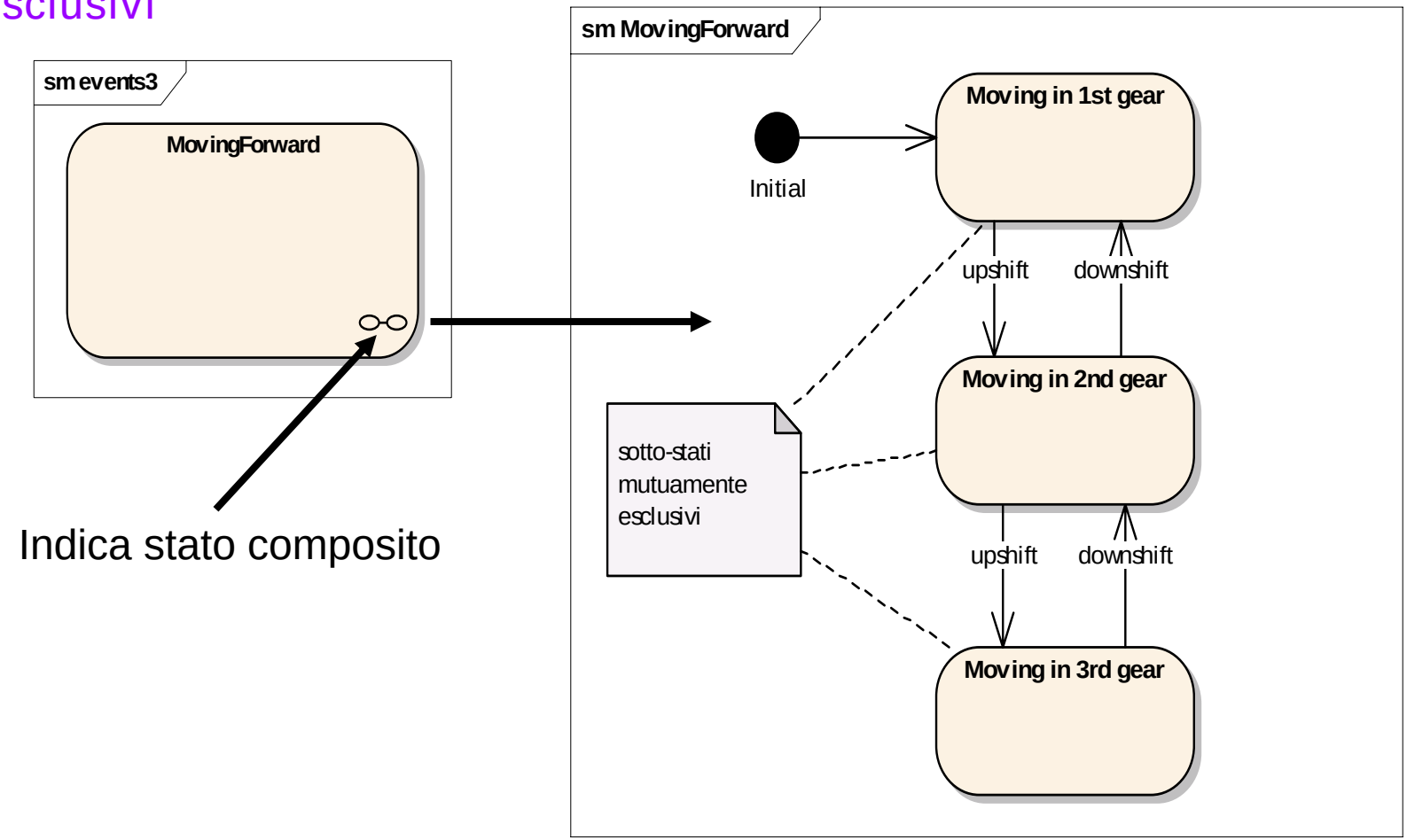
Relazione tra diagrammi di stato e diagrammi di sequenza 4/4

- Il diagramma a stati completo (relativo ai due scenari discussi)



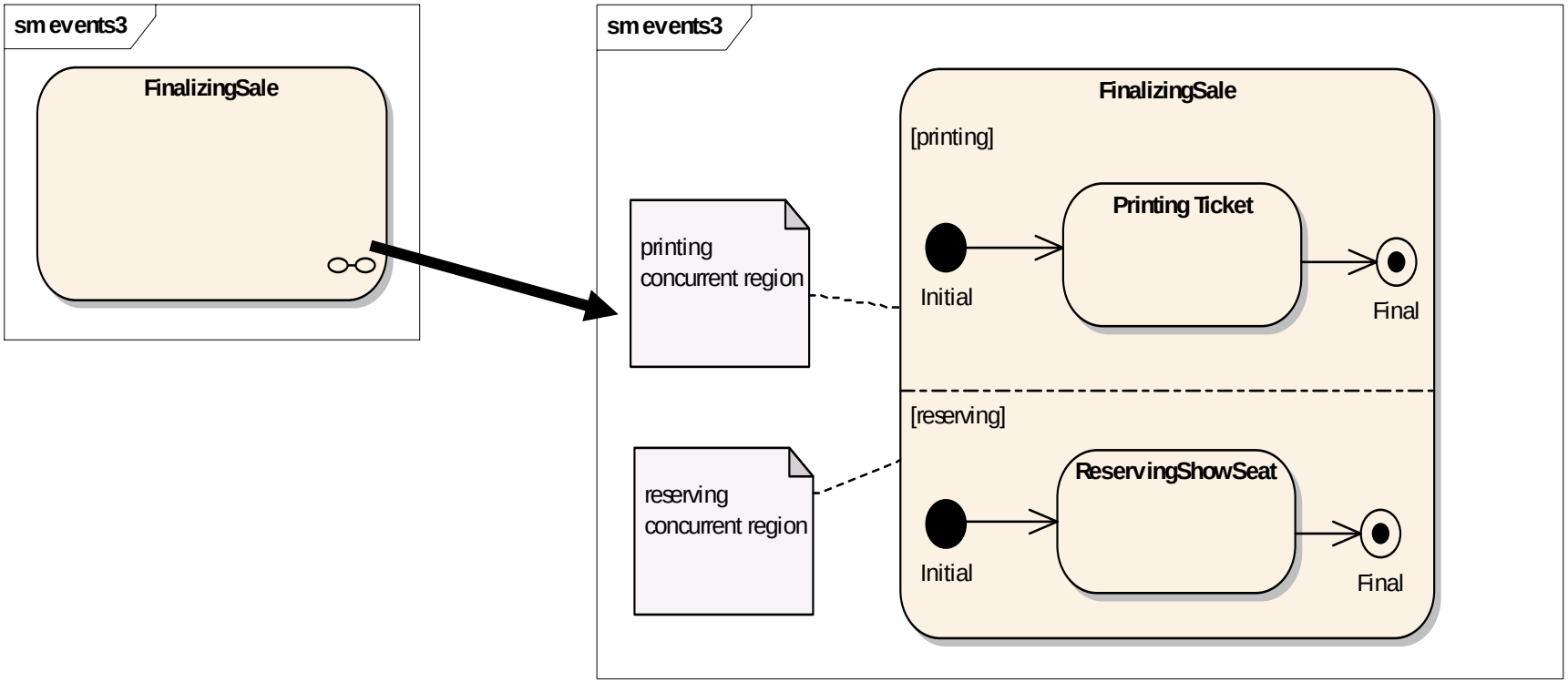
Diagrammi di stato compositi 1/2

- Uno stato può contenere al suo interno più sottostati mutuamente esclusivi



Diagrammi di stato compositi 2/2

- Uno stato può contenere al suo interno sottostati concorrenti



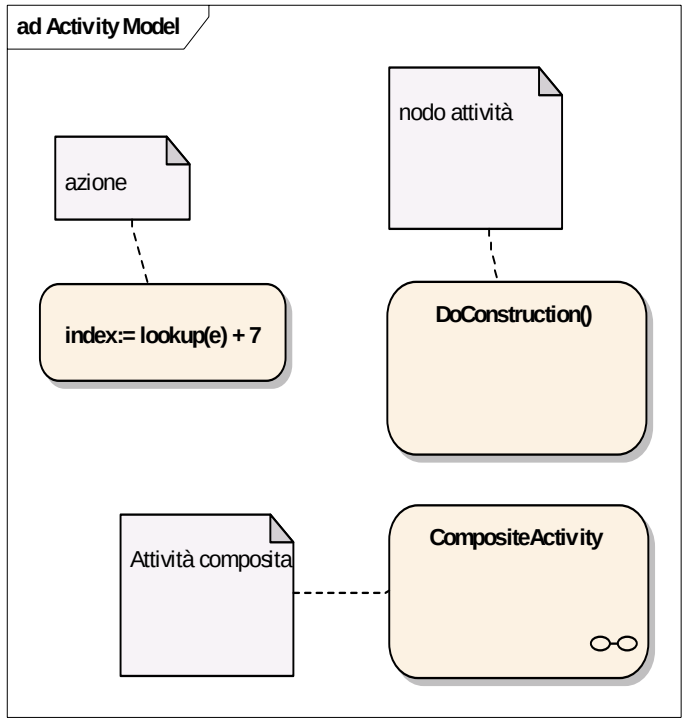
- Introduzione: approccio e motivazioni
- Modellare la struttura
- **Modellare la dinamica**
  - Diagrammi di sequenza
  - Diagrammi di stato (statechart)
  - Diagrammi di attività
  - Diagrammi dei casi d'uso
  - Diagrammi delle collaborazioni
  - Timing diagram

Diagrammi di attività

- Un diagramma di attività mostra il flusso di azioni relativo ad un'attività
- Un'attività è un'esecuzione non atomica di operazioni all'interno di una macchina a stati
- L'esecuzione di un'attività viene decomposta in azioni atomiche
- Ogni azione può o meno cambiare lo stato del sistema
- I diagrammi di attività sono spesso usati per descrivere la logica di un algoritmo (sono l'equivalente UML dei diagrammi di flusso)
- Graficamente un diagramma di attività è un insieme di archi e nodi

Diagramma di attività – azioni e nodi attività

- Azioni (atomiche)
  - Valutazione di espressioni
  - Assegnamenti / Ritorno di un valore
  - Invocazione di un'operazione su un oggetto
  - Creazione/distruzione di un oggetto



- Nodi Attività
  - Raggruppamento di azioni atomiche o di altri nodi attività
  - Un'azione può essere vista come un'attività che non può essere ulteriormente decomposta
  - Espandendo un nodo attività si ottiene un altro diagramma di attività (attività composita)
  - A parte questa differenza, i due concetti sono rappresentati mediante lo stesso simbolo grafico

Diagramma di attività – Flusso di controllo

- Quando un'azione o un'attività viene completata, il flusso di controllo passa al nodo azione (attività) immediatamente successivo
- Il flusso di controllo viene specificato mediante frecce che collegano due nodi (attività o azione)
- Il flusso mostrato in figura è quello più semplice: il flusso sequenziale

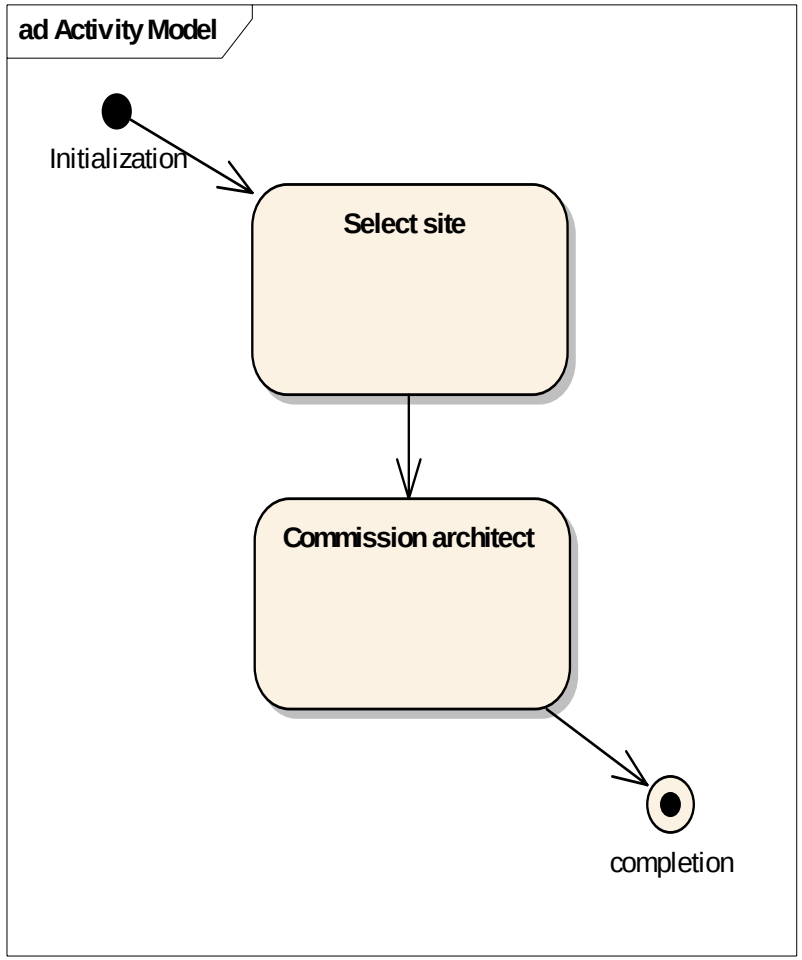


Diagramma di attività – Branch e merge

- Un altro tipo di flusso possibile è il branch
- Un branch è rappresentanto da un diamante
- Ogni branch ha:
  - Un flusso entrante
  - Due o più flussi uscenti
  - Una condizione logica (talvolta implicita) che determina quale dei flussi uscenti verrà eseguito da una particolare esecuzione
- Quando due flussi si riuniscono, è possibile usare ancora il simbolo di diamante; in questo caso viene detto merge
- Ogni merge ha almeno due flussi entranti e un flusso uscente

Diagramma di attività – Branch e merge

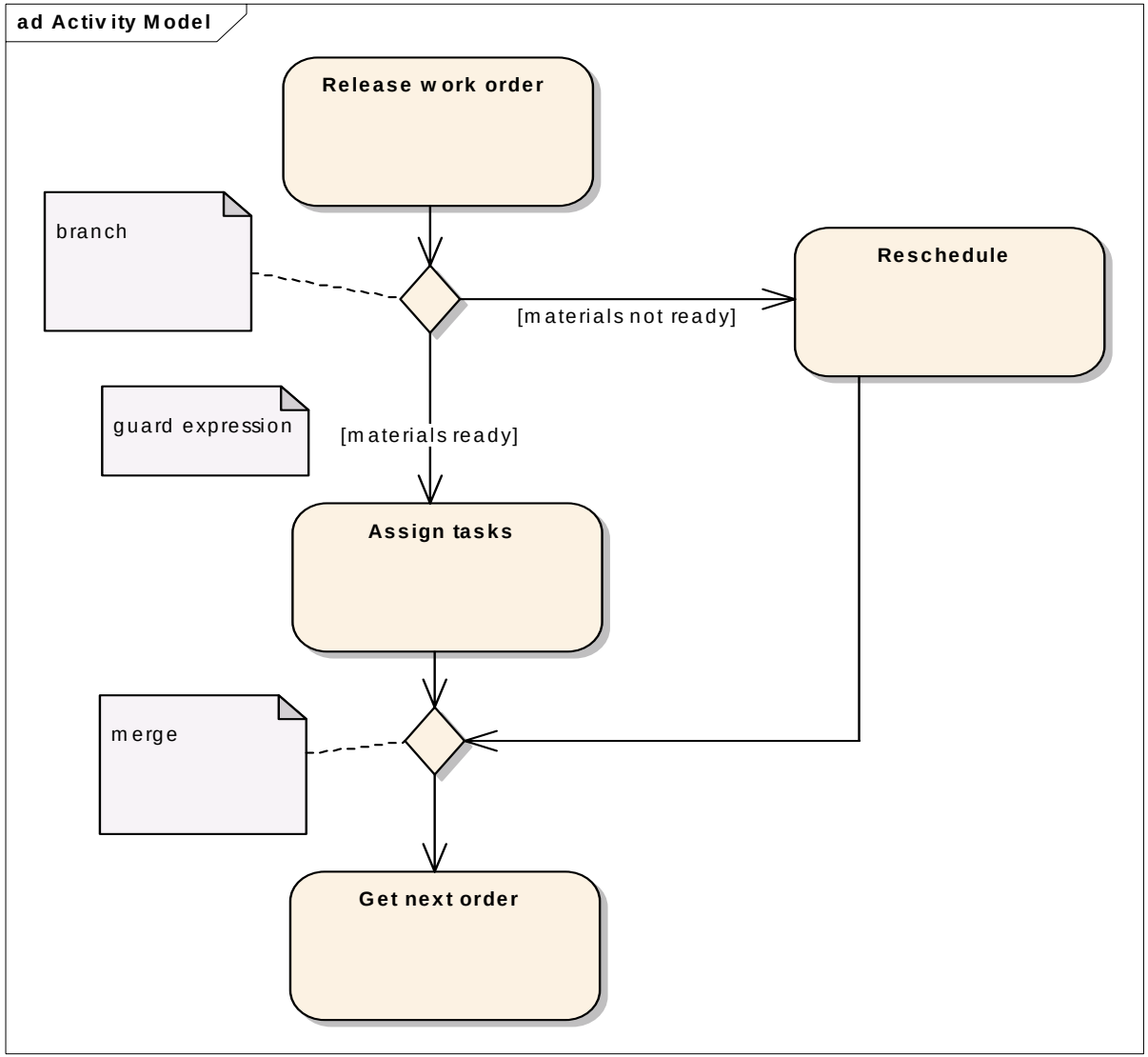


Diagramma di attività – Fork e Join

- Alcuni flussi possono essere concorrenti
- In UML vengono usate delle barre di sincronizzazione per specificare fork e join di flussi di controllo paralleli
- Un join rappresenta la sincronizzazione di due o più flussi di controllo concorrenti
- Un join ha due o più flussi entranti e un flusso uscente
- LA sincronizzazione sul join attende che tutte le attività nei flussi entranti abbiamo terminato la loro esecuzione, prima di procedere
- Join e fork si devono bilanciare
- Le attività in un flusso di controllo parallelo comunicano tra loro spedendosi segnali (stile di comunicazione detto coroutine)

Diagramma di attività – Fork e Join

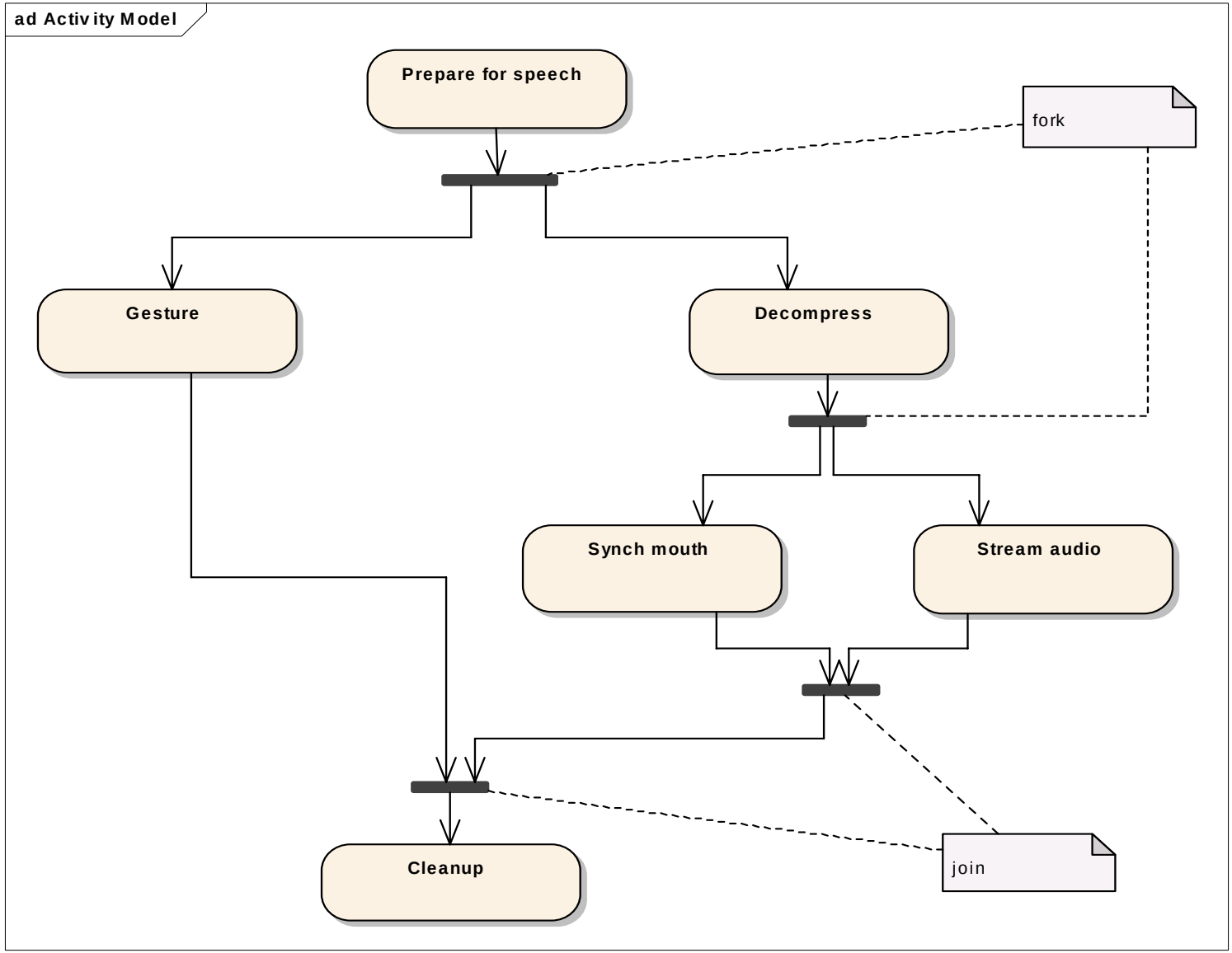


Diagramma di attività – Swimlane

- A volte è utile partizionare le attività in base alle entità coinvolte che le devono svolgere
- In UML si usano a tale scopo le cosiddette swimlane
- Una swimlane è un raggruppamento (verticale oppure orizzontale) di attività eseguite da una stessa entità (ad esempio, una classe)
- Ogni swimlane deve avere un nome univoco nel diagramma
- Le swimlane rappresentano responsabilità specifiche nel contesto di un'attività generale
- Le attività sono associate univacamente ad un'unica swimlane
- Solo le transizioni (flussi) possono attraversare due o più swimlane

Diagramma di attività – Swimlanes

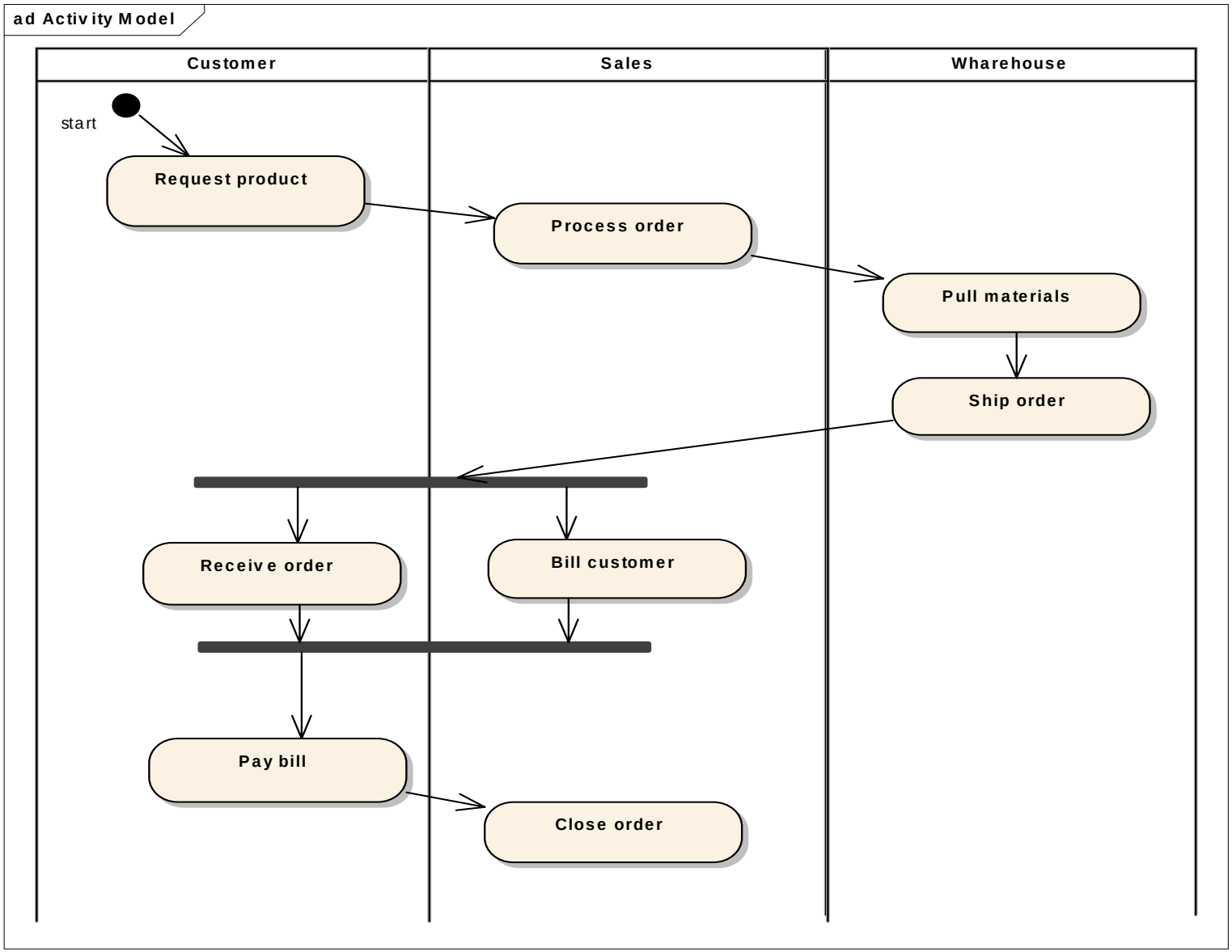


Diagramma di attività – Flusso di oggetti

- A volte è utile evidenziare non solo il flusso di controllo, ma anche gli oggetti coinvolti
- Un'attività può creare un oggetto
- Un'altra attività può contenere azioni che modificano lo stato interno di un oggetto
- Il flusso del valore (stato) di un oggetto tra due azioni è detto flusso dell'oggetto
- Lo stato viene rappresentato tra parentesi quadre all'interno dell'oggetto, oppure come constraint in una nota associata all'oggetto stesso

Diagramma di attività – Flusso di oggetti