Introduzione a UML

Perché modelliamo

Un modello è una semplificazione della realtà

I ***modelli***

- ci aiutano a “visualizzare” un sistema *come è* o *come vorremmo che fosse*
- ci permettono di specificare la struttura o il comportamento di un sistema
- ci forniscono un “template” che ci guida nella costruzione di un sistema
- documentano le decisioni che abbiamo preso

Perché modelliamo

- *Divide et impera*: tramite i modelli ci focalizziamo su un solo aspetto alla volta
- ogni modello può essere espresso a differenti livelli di precisione
- per un sistema non banale:
  - ⇒ non un solo modello
  - ⇒ ma un piccolo insieme di modelli, che possono essere costruiti e studiati separatamente, ma che sono strettamente interrelati

Approcci metodologici

Un po' di storia ….

anni '70	***Approccio Strutturato***
anni '80	***Information Engineering***
anni '90	***Approccio Object Oriented***

Non si tratta di vere e proprie metodologie, ma:

- sono impostazioni globali, che coprono più fasi del ciclo di sviluppo / manutenzione
- hanno avuto (ed hanno) notevole diffusione
- hanno ispirato (ed ispirano) le metodologie più diffuse

Approccio Strutturato

(Yourdon - De Marco)

Caratteristiche:

- "primo" tentativo di fornire regole per le attività di sviluppo SW
- il concetto di base è la modularizzazione
- utilizzo di modelli formali e diagrammatici
- utilizzo di tecniche specifiche per le diverse fasi di sviluppo
- l'attenzione è rivolta soprattutto alla componente funzionale

Approccio Strutturato

Limiti

- origine anni ‘70: è fondato sul paradigma funzionale (process driven) caratteristico di sistemi mainframe batch
- tecniche e modelli diversi per dati e processi, per analisi e disegno
- difficoltà nel “passaggio” dall’analisi al disegno

Che cosa resta attuale dell’approccio strutturato?

- la distinzione ferrea tra analisi (***cosa***) e disegno (***come***)
- l’attenzione alle ***interazioni tra sistema e ambiente esterno***
- alcune ***tecniche di analisi*** (DFD, STD), sia pur calate in un diverso contesto metodologico
- i principi guida del disegno strutturato (***coesione***, ***coupling***)

Information Engineering

Inizio anni ‘80 (Clive Finkelstein, James Martin)

Caratteristiche:

- utilizzo estensivo di modelli formali e diagrammatici
- automazione della produzione del software (CASE, generatori di codice)
- attenzione rivolta principalmente ai dati

Information Engineering

Limiti

- origine anni ‘80: approccio data driven, ma ancora legato alla separazione dati - processi
- tecniche di analisi e disegno derivate dall’approccio strutturato
- l’ambiente target previsto è ancora monopiattaforma

Che cosa resta attuale dell’Information Engineering?

- la suddivisione in macrofasi
- l’attenzione all’automazione del processo di sviluppo e manutenzione (CASE …)
- l’insistenza sulla partecipazione dell’utente a tutte le fasi del ciclo di sviluppo / manutenzione

Approccio Object Oriented

Inizio anni ‘70	**Programmazione OO**
fine anni ‘80	**Analisi** e **Disegno OO**

Caratteristiche:

- superamento della distinzione tra dati e funzioni
- concetti “nuovi”: ereditarietà, information hiding, …
- forte tendenza al riutilizzo di componenti software già definite

Metodi di analisi e disegno OO

Esplosione dei metodi:

- dal 1989 al 1994 sono passati da 5 a oltre 50, ma…
- con differenze spesso solo superficiali - notazioni, terminologia e poco più

- La confusione esistente a livello metodologico si ripercuote a livello di strumenti CASE per la progettazione object oriented

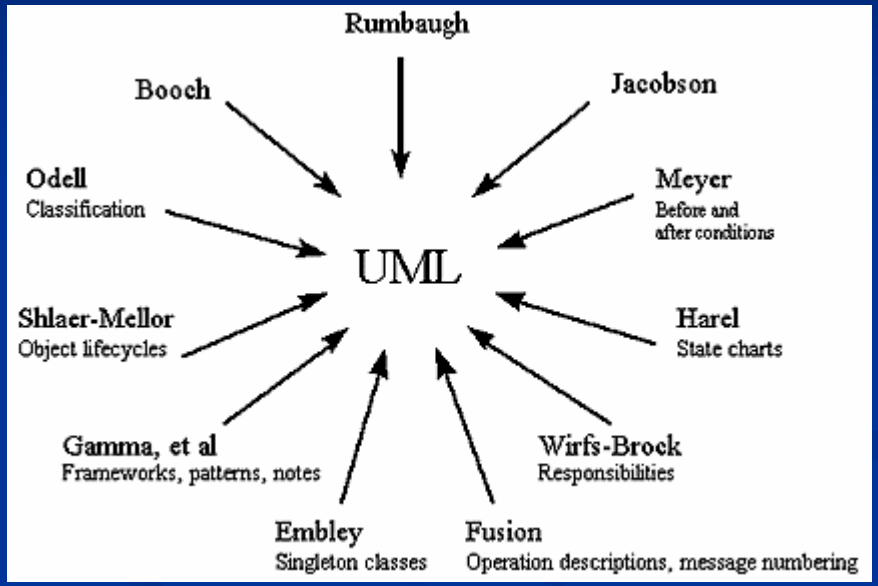
Convergenza dei metodi: *Unified Modeling Language*

Unified Modeling Language

- un linguaggio (e notazione) universale, per la creazione di modelli software
- nel novembre ‘97 è diventato uno **standard** approvato dall’**OMG** (Object Management Group)
- numerosi i co-proponenti: Microsoft, IBM, Oracle, HP, Platinum, Sterling, Unysis …..

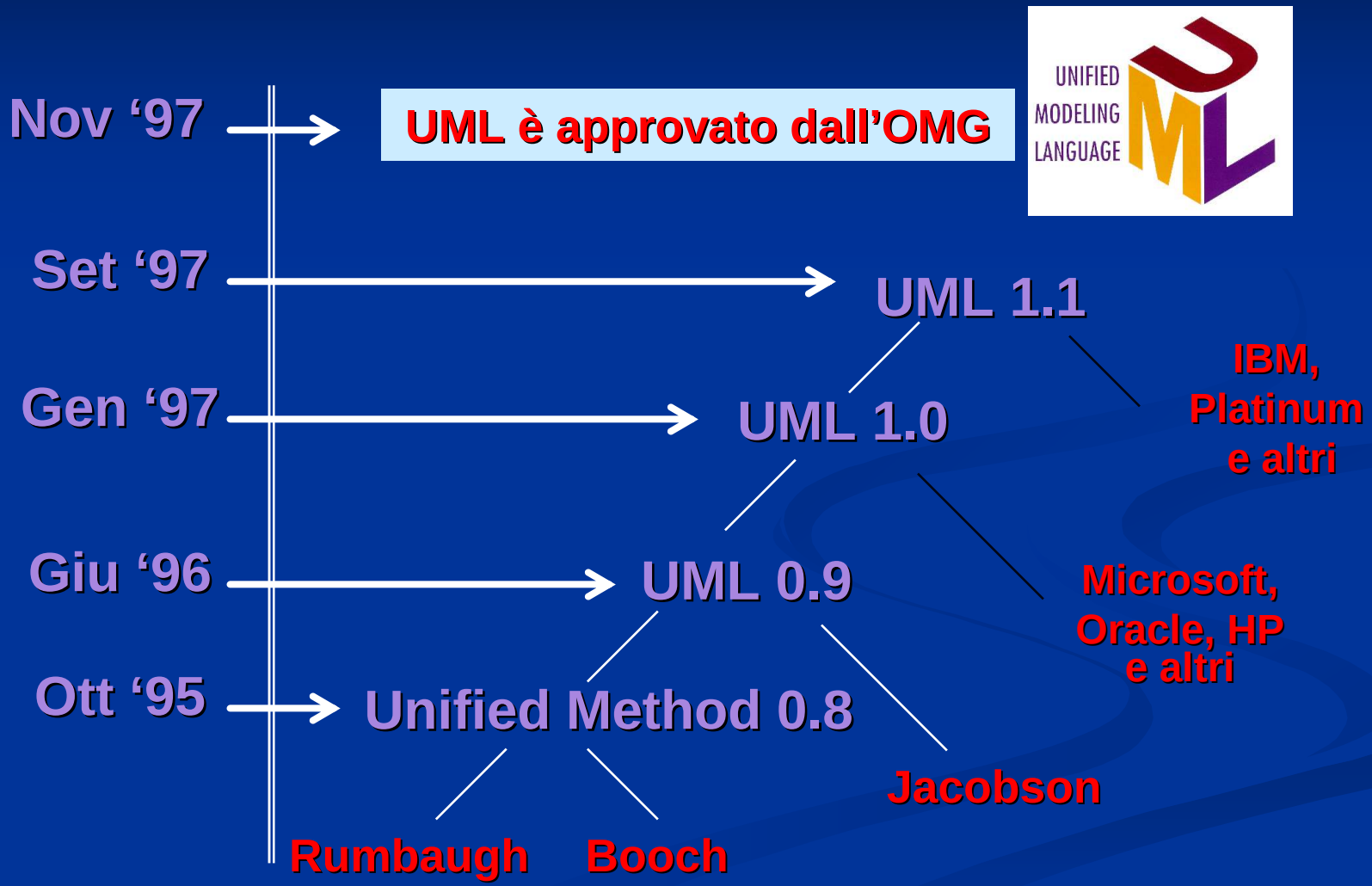
Unified Modeling Language

- è l'unificazione dei metodi:
  - Booch-93 di **Grady Booch**
  - OMT di **Jim Rumbaugh**
  - OOSE di **Ivar Jacobson**



- ha accolto inoltre le idee di numerosi altri metodologi
- è tuttavia indipendente dai metodi, dalle tecnologie, dai produttori

Storia di UML



UML - linguaggio universale

- **linguaggio** per specificare, costruire, visualizzare e documentare gli artefatti di un sistema

- **universale**: può rappresentare sistemi molto diversi, da quelli web ai legacy, dalle tradizionali applicazioni Cobol a quelle object oriented e a componenti

UML non è un metodo

- è un **linguaggio di modellazione**, non un metodo, né una metodologia
- definisce una notazione standard, basata su un meta-modello integrato degli “elementi” che compongono un sistema software
- non prescrive una sequenza di processo, cioè non dice “prima bisogna fare questa attività, poi quest’altra”

UML e Processo Software

"un unico processo universale buono per tutti gli stili dello sviluppo non sembra possibile e tanto meno desiderabile"

- in realtà UML assume un processo:
  - **basato sui Casi d'Uso (*use case driven*)**
  - **incentrato sull'architettura**
  - **iterativo e incrementale**
- i dettagli di questo processo di tipo generale vanno adattati alle peculiarità della cultura dello sviluppo o del dominio applicativo di ciascuna organizzazione

UML: meta-modello

- UML si fonda su un meta-modello integrato, che definisce le caratteristiche e le relazioni esistenti tra i diversi elementi (classi, attributi, moduli, …)
- Il meta-modello è la base per l'implementazione dell'UML da parte dei produttori di strumenti di sviluppo (CASE, ambienti visuali, …) e per l'interoperabilità tra i diversi strumenti

UML: meta-modello

- UML prevede una serie di **modelli diagrammatici** basati sul meta-modello
- gli elementi del meta-modello possono comparire in diagrammi di diverso tipo
- alcuni elementi (ad es. la “classe”) hanno una icona che li rappresenta graficamente

Diagrammi UML

livello "logico":

dei casi d'uso - **Use Case Diagram**
delle classi - **Class Diagram**
di sequenza - **Sequence Diagram**
di collaborazione - **Collaboration Diagram**
di transizione di stato - **Statechart Diagram**
delle attività - **Activity Diagram**

livello "fisico":

dei componenti - **Component Diagram**
di distribuzione dei componenti - **Deployment Diagram**

Diagramma dei casi d’uso

Mostra:

- le modalità di utilizzo del sistema (casi d’uso)
- gli utilizzatori e coloro che interagiscono con il sistema (attori)
- le relazioni tra attori e casi d’uso

Un **caso d’uso**

- rappresenta un possibile “modo” di utilizzo del sistema
- descrive l’interazione tra attori e sistema, non la “logica interna” della funzione

una funzionalità dal punto di vista di chi la utilizza

Diagramma dei casi d’uso

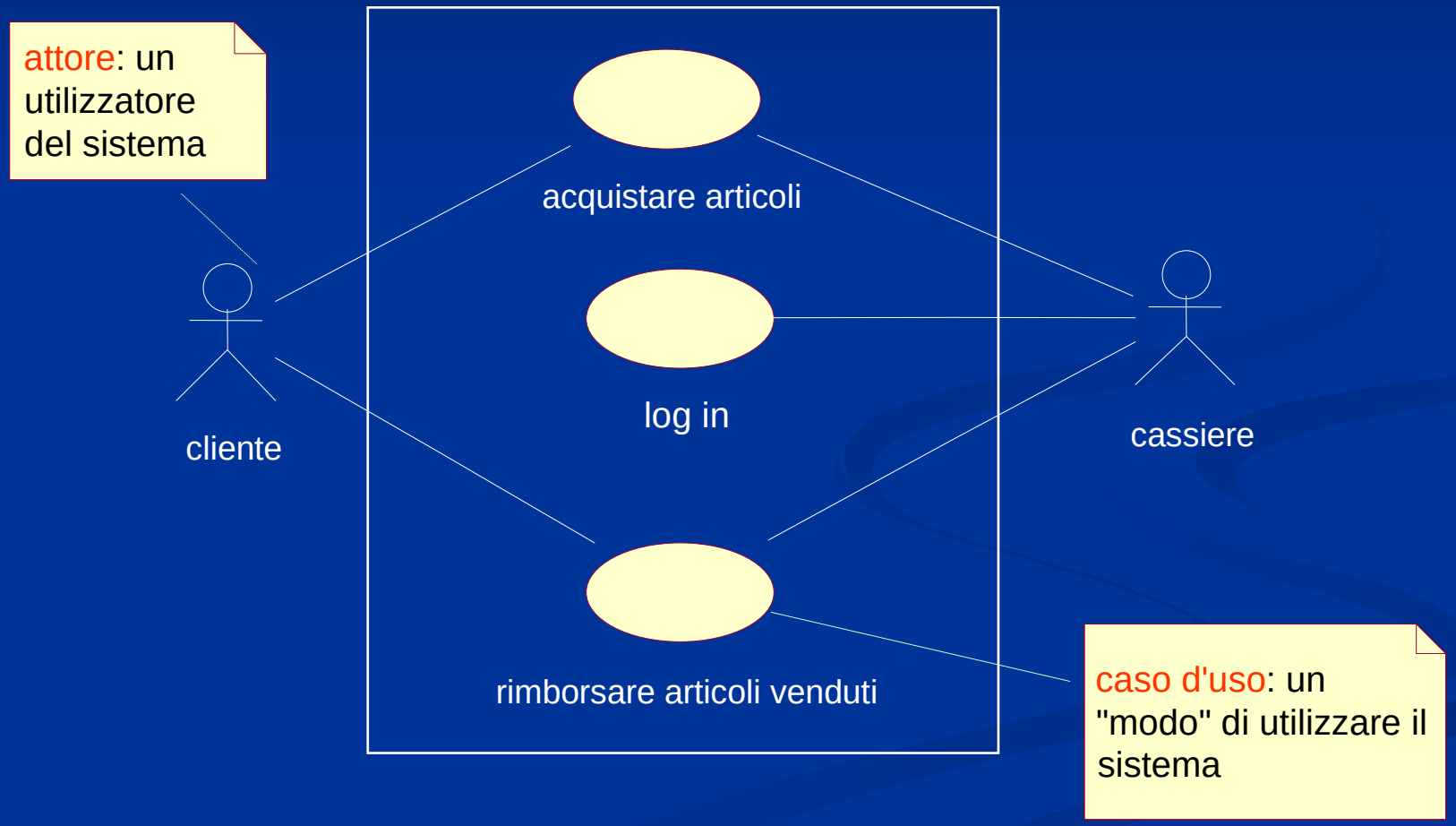


Diagramma delle classi

- è il caposaldo dell'object oriented
- rappresenta le classi di oggetti del sistema con i loro attributi e operazioni
- mostra le relazioni tra le classi (associazioni, aggregazioni e gerarchie di specializzazione/generalizzazione)
- può essere utilizzato a diversi livelli di dettaglio (in analisi e in disegno)

Diagramma delle classi

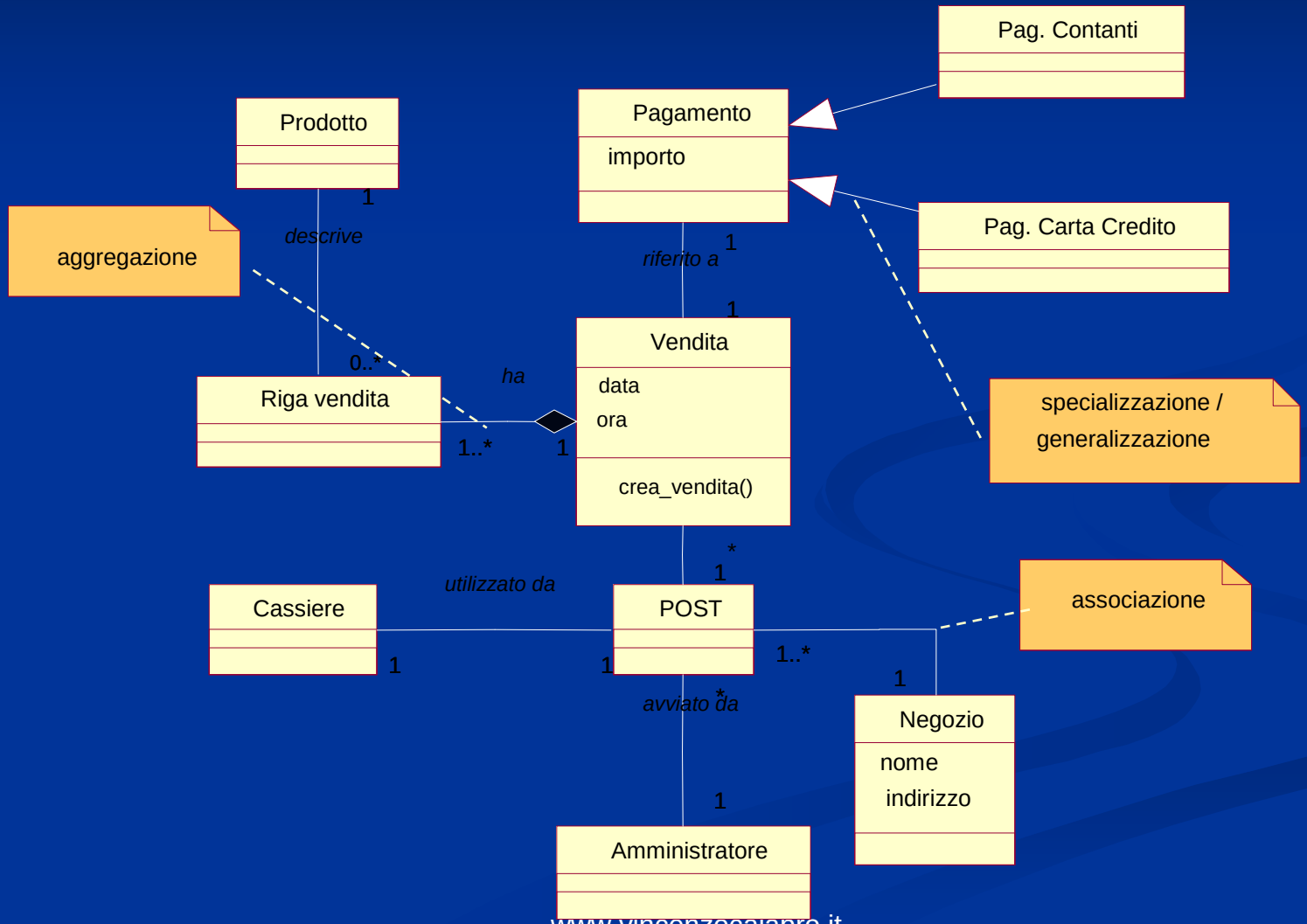


Diagramma di sequenza

- è utilizzato per definire la logica di uno scenario (specifica sequenza di eventi) di un caso d'uso (in analisi e poi ad un maggior livello di dettaglio in disegno)
- è uno dei principali input per l'implementazione dello scenario
- mostra gli oggetti coinvolti specificando la sequenza temporale dei messaggi che gli oggetti si scambiano
- è un **diagramma di interazione**: evidenzia come un caso d'uso è realizzato tramite la collaborazione di un insieme di oggetti

Diagramma di sequenza

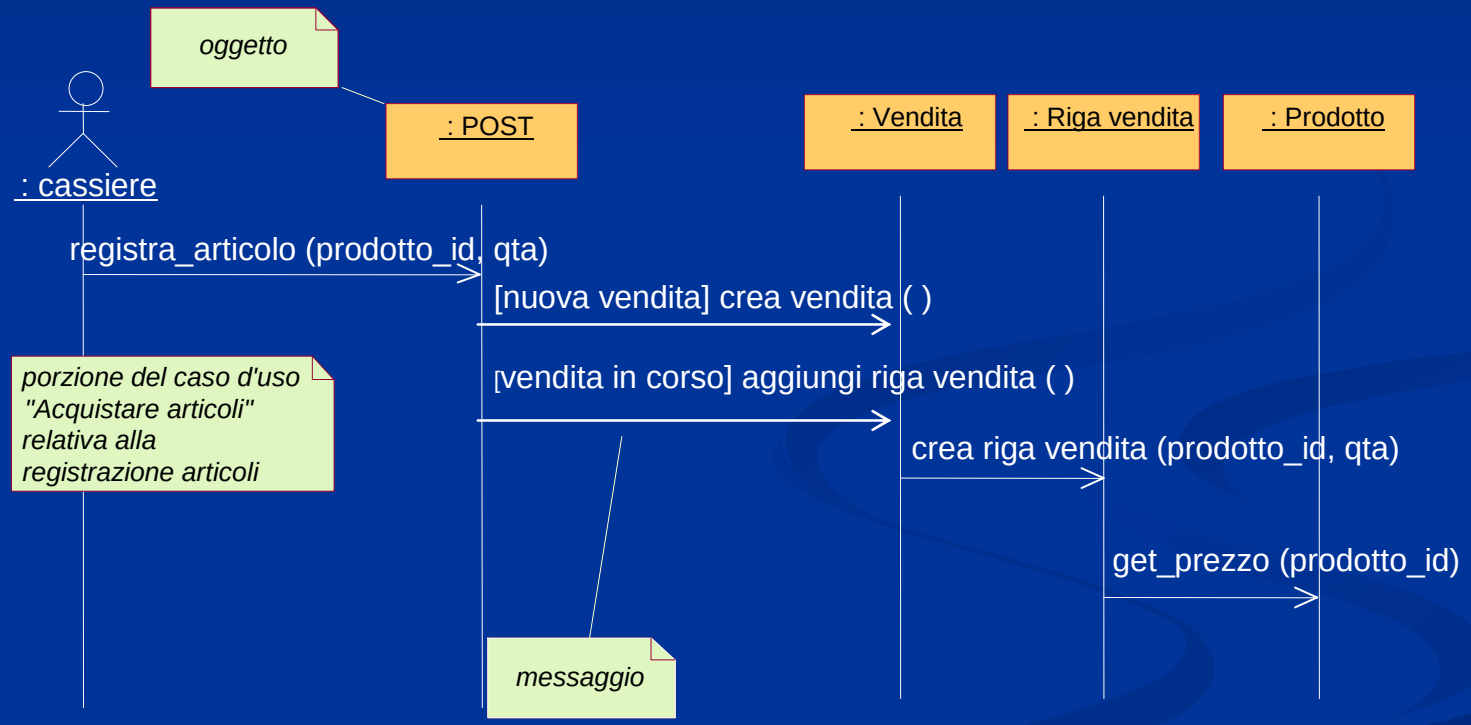


Diagramma di collaborazione

- è un **diagramma di interazione**: rappresenta un insieme di oggetti che collaborano per realizzare il comportamento di uno scenario di un caso d'uso
- a differenza del diagramma di sequenza, mostra i link (legami) tra gli oggetti che si scambiano messaggi, mentre la sequenza di tali messaggi è meno evidente
- può essere utilizzato in fasi diverse (analisi, disegno di dettaglio)

Diagramma di collaborazione

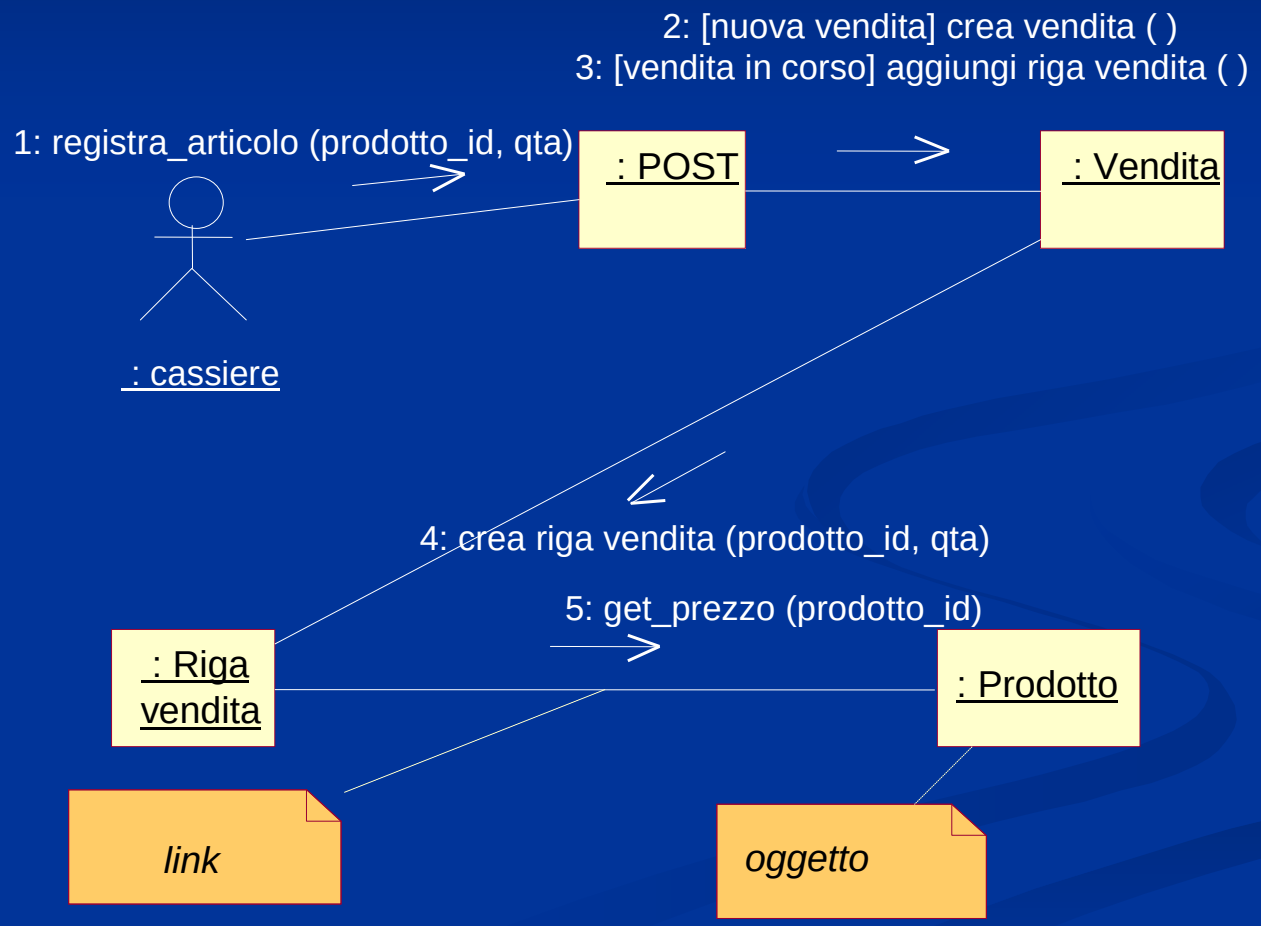


Diagramma transizioni di stato

- è normalmente utilizzato per modellare il ciclo di vita degli oggetti di una singola classe
- mostra gli eventi che causano la transizione da uno stato all'altro, le azioni eseguite a fronte di un determinato evento
- quando un oggetto si trova in un certo stato può essere interessato da determinati eventi (e non da altri)
- è opportuno utilizzarlo solo per le classi che presentano un ciclo di vita complesso e segnato da una successione ben definita di eventi

Diagramma transizioni di stato

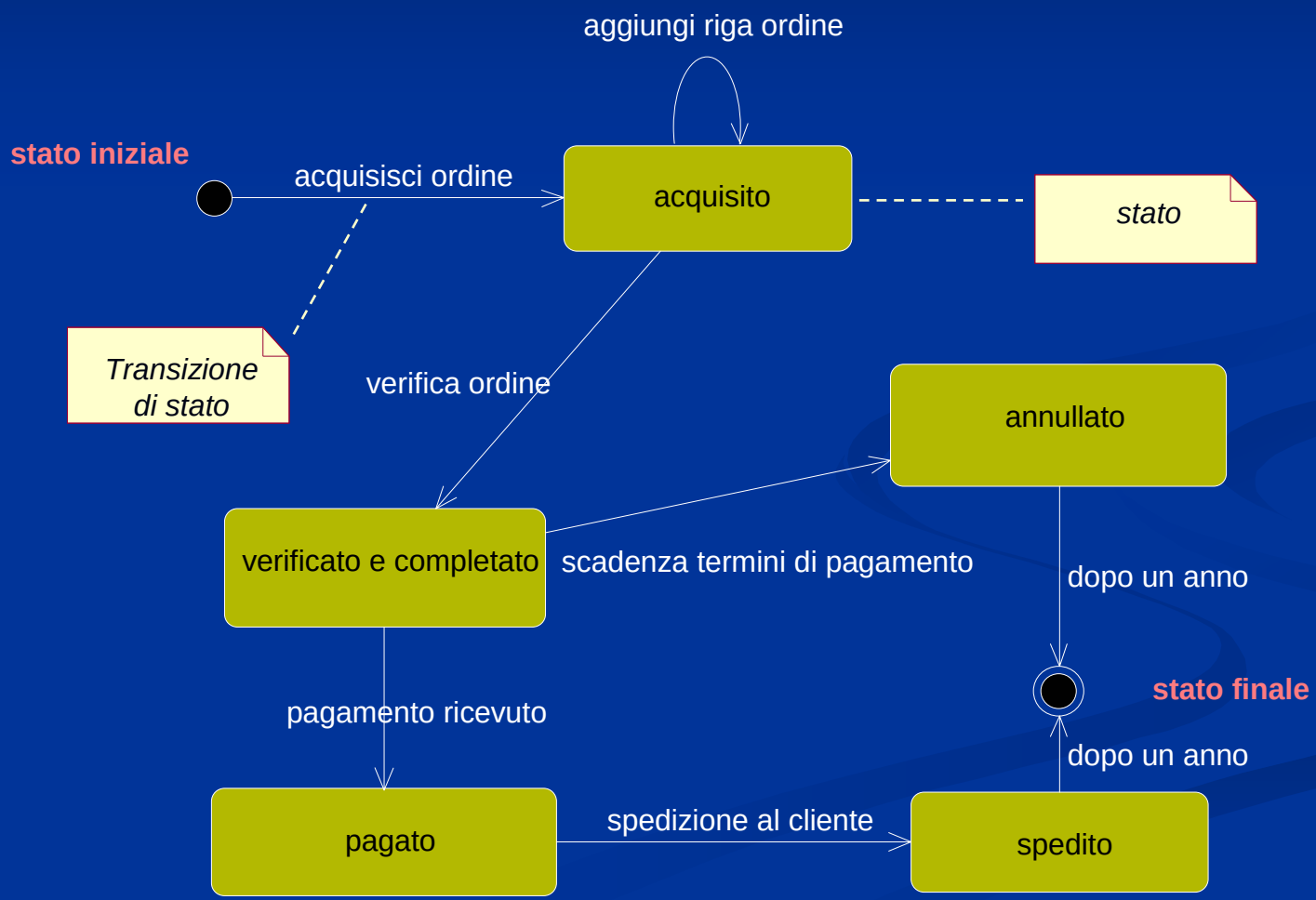


Diagramma di attività

- rappresenta sistemi di workflow, oppure la logica interna di un processo (processo di business o processo di dettaglio), di un caso d'uso o di una specifica operazione di una classe
- permette di modellare processi paralleli e la loro sincronizzazione
- è un caso particolare di diagrammi di stato, in cui ogni stato è uno stato di attività

Diagramma di attività

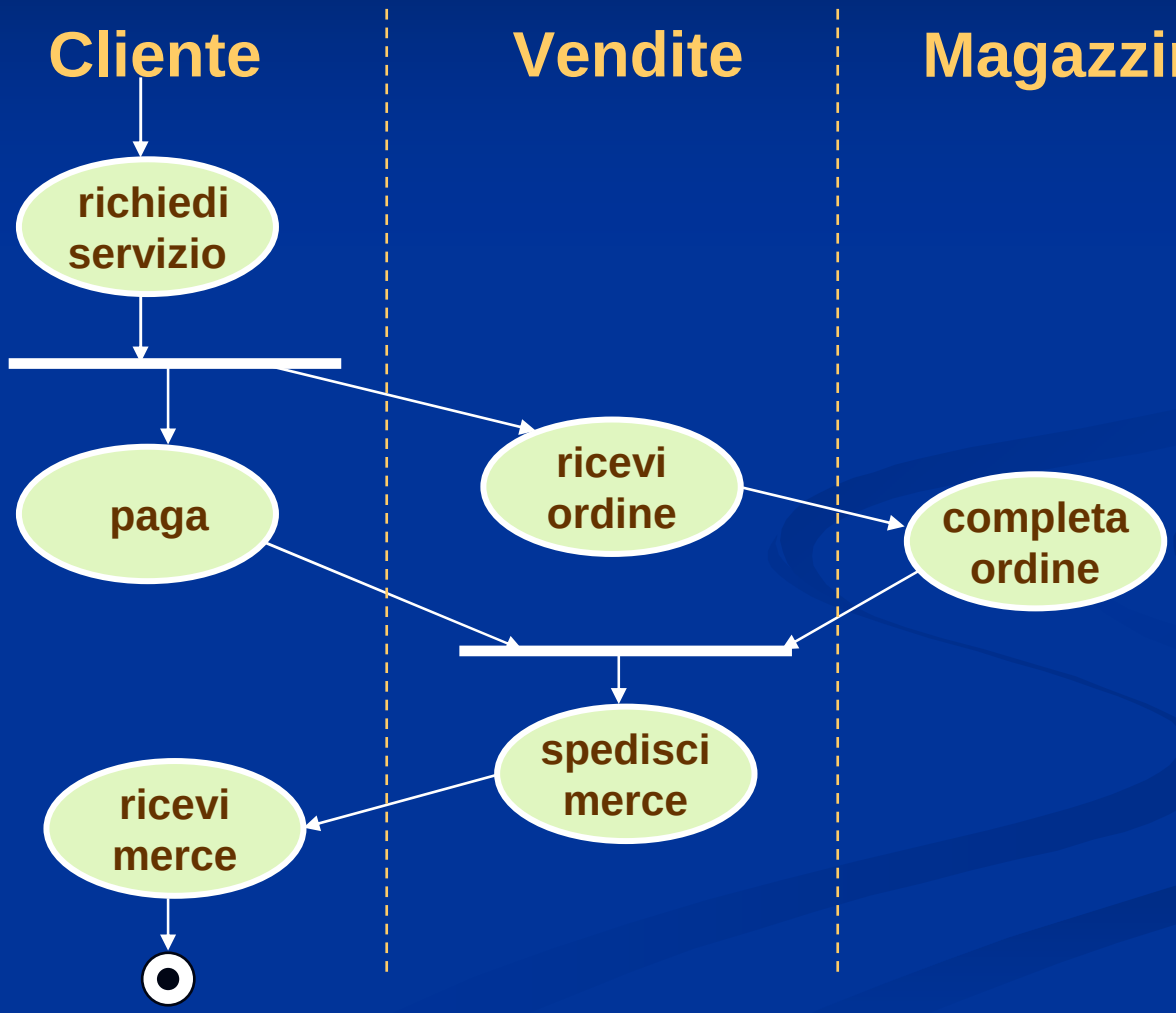


Diagramma dei componenti

- evidenzia l'organizzazione e le dipendenze tra i componenti software
- i componenti (come a livello logico i casi d’uso o le classi) possono essere raggruppati in package
    - *un componente è una qualunque porzione fisica riutilizzabile con un’identità e un’interfaccia (dichiarazione di servizi offerti) ben definite*
    - *un componente può essere costituito dall’aggregazione di altri componenti*

Diagramma dei componenti

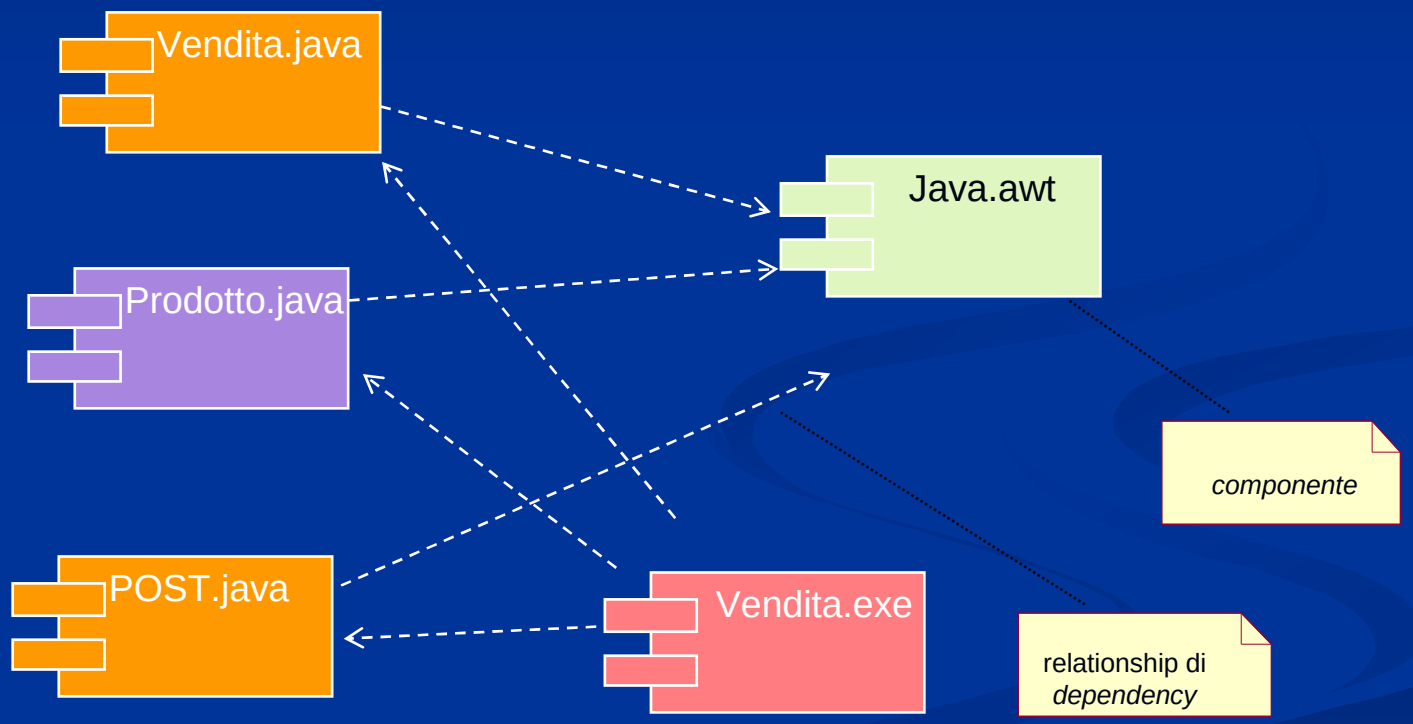
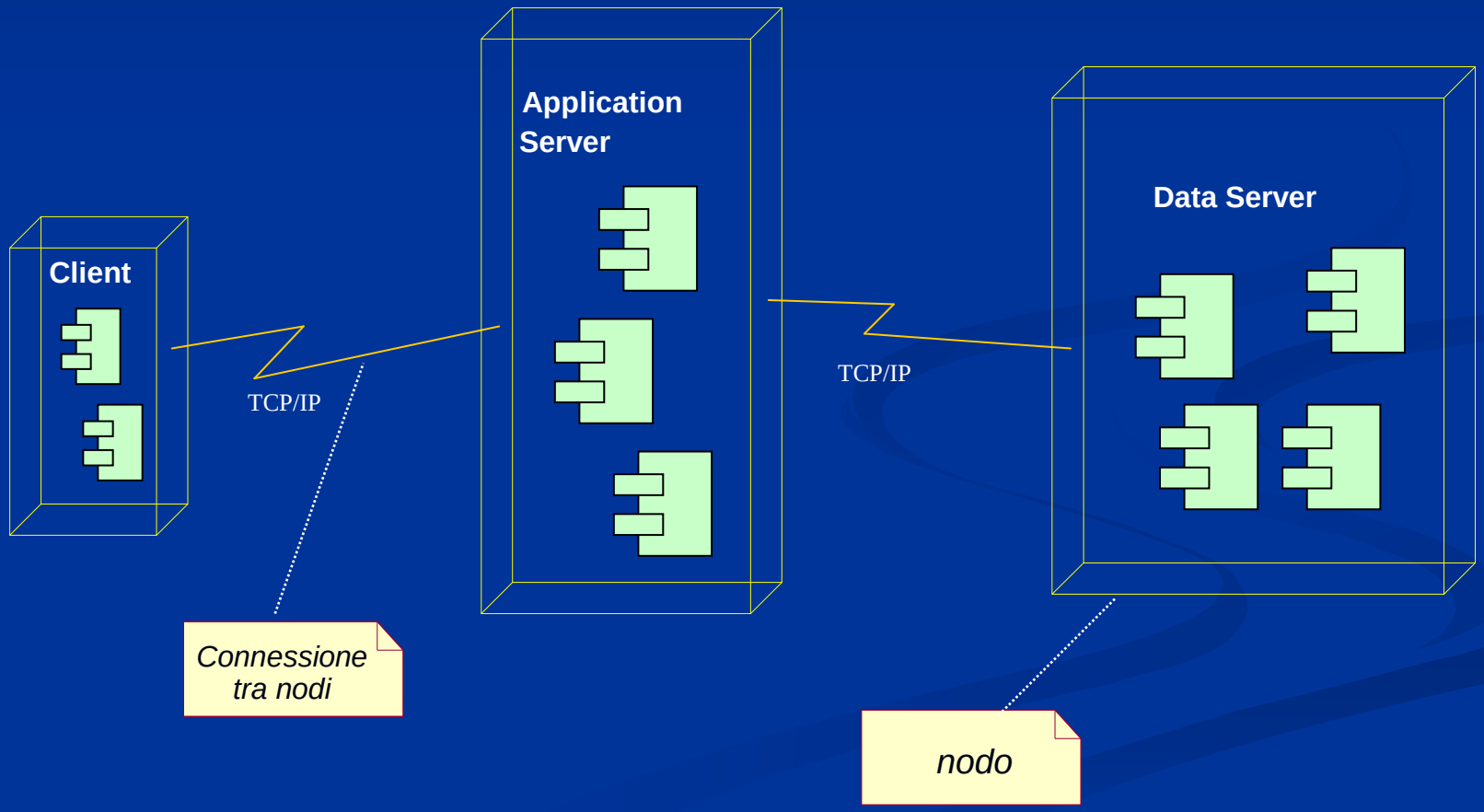


Diagramma di distribuzione

- è utilizzato per mostrare come sono configurate e allocate le unità hardware e software per un’applicazione

- evidenzia la configurazione dei nodi elaborativi in ambiente di esecuzione (run-time), e dei componenti, processi ed oggetti allocati su questi nodi

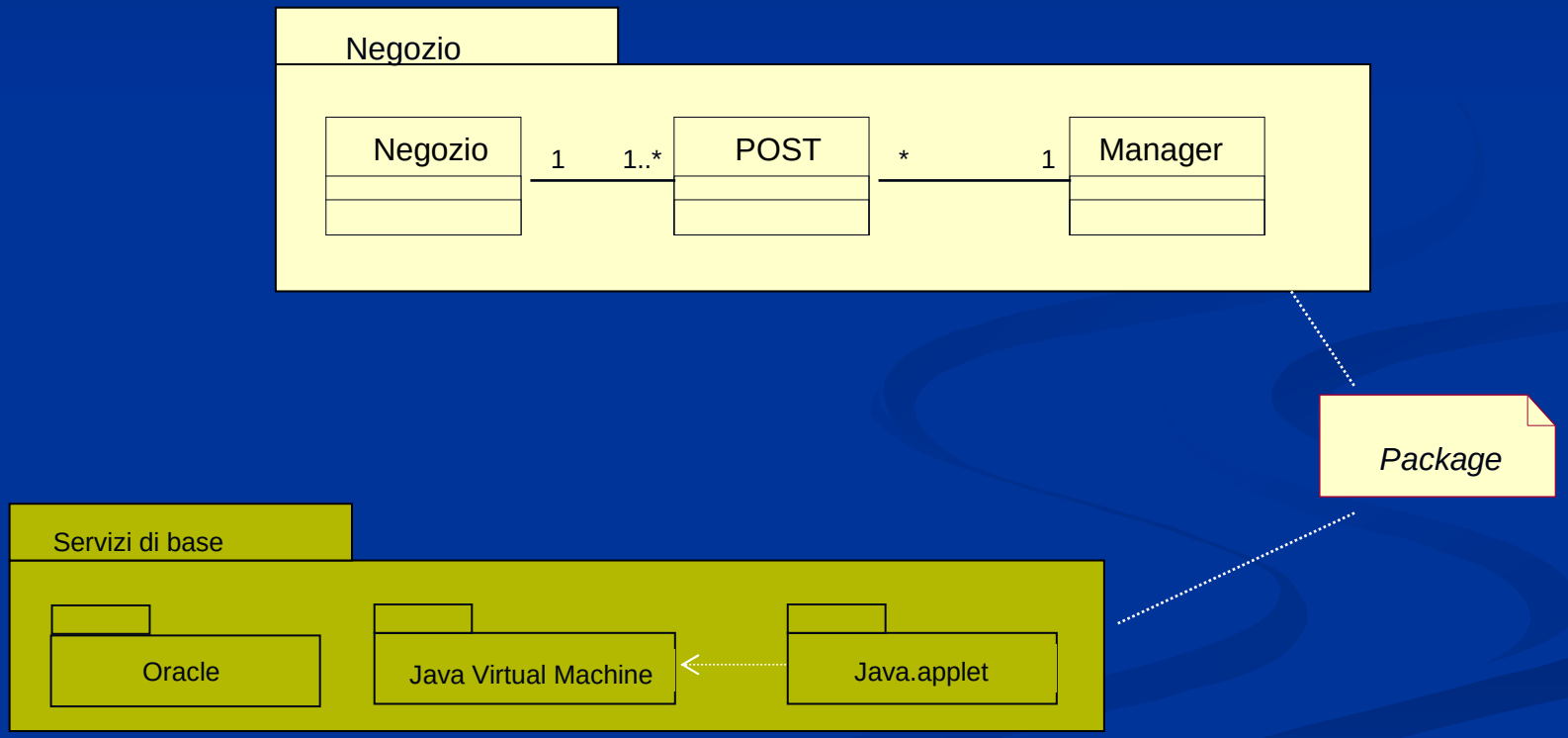
Diagramma di distribuzione



Package

- consente di partizionare il sistema in sottosistemi costituiti da elementi omogenei di:
  - natura **logica** (classi, casi d'uso, …)
  - natura **fisica** (moduli, tabelle, …)
  - altra natura (processori, risorse di rete, …)
- ogni elemento appartiene ad un solo package
- un package può referenziare elementi appartenenti ad altri package

Package


Negozio
Negozio
1
1..*
POST
*
1
Manager
Package
Servizi di base
Oracle
Java Virtual Machine
Java.applet

UML - considerazioni finali

- UML rappresenta un'evoluzione dei modelli preesistenti, più che una rivoluzione
- è adatto a esprimere modelli di varia tipologia, creati per obiettivi diversi
- può descrivere un sistema software a diversi livelli di astrazione, dal piano più svincolato dalle caratteristiche tecnologiche fino all'allocazione dei componenti software nei diversi processori in un'architettura distribuita

UML - considerazioni finali

UML è sufficientemente complesso per rispondere a tutte le necessità di modellazione, ma è opportuno “ritagliarlo” in base alle specifiche esigenze dei progettisti e dei progetti, utilizzando solo ciò che serve nello specifico contesto

“keep the process as simple as possible!”

Grazie