Utilità dell'UML

Corso Serale

Progettazione del Sofware

Perché Modellare un Sistema

- Necessità di realizzare un "artefatto", indipendentemente dalla sua dimensione e settore di interesse (una casa, un particolare macchinario, un ponte o un dipartimento di un'azienda.. Un software!)
- Obiettivo: produrre una versione razionalizzata del sistema reale, al fine di evidenziarne l'aspetto finale, le prestazioni, l'affidabilità, il comportamento, e così via.

Perché modellare usando UML

- I modelli vengono particolarmente apprezzati quando ci si trova ad affrontare sistemi vasti e/o molto articolati.
- Problema: Comprendere la complessità nella sua interezza
- Necessità di avvalersi di modelli in grado di descrivere, in maniera più semplice, sistemi comunque complessi.
- Grazie alle “viste” UML la mente umana riesce a concentrarsi, in ogni istante, su un numero limitato di aspetti del sistema: quelli ritenuti importanti per quel particolare momento (vista)

Infine…

- I modelli sono molto importanti perché permettono, con la assistenza dei clienti, di definire i requisiti del sistema e, con la cooperazione del proprio gruppo, di razionalizzarne il processo di sviluppo.

In breve

- analisi dei tempi
- stima dei costi
- il piano dell'allocazione delle risorse
- la distribuzione del carico di lavoro.
- risparmiare tempo e quindi denaro
- ridurre i fattori di rischio presenti in ogni progetto
- studiare la risposta del sistema a particolari sollecitazioni

Qualità di un modello

- Proprietà peculiari che il modello risultante deve possedere; deve essere:
- **accurato:** deve descrivere correttamente il sistema da realizzare;
- **consistente**: le diverse viste devono completarsi vicendevolmente ed essere prive di incoerenze;
- **semplice**: deve poter essere compreso, senza troppi problemi, da persone estranee al processo di modellazione;
- **manutenibile**: la variazione dello stesso deve essere il più semplice possibile;

Aree di intervento.

Principalmente, l'UML può essere utilizzato per:

- visualizzare;
- specificare;
- costruire;
- documentare;

un qualsiasi artefatto ingegneristico, ed in particolare, **un progetto software.**

SWENG

- Il “mapping” tra modello e linguaggio di programmazione permette la realizzazione di funzioni di “reverse engeneering”: fornendo ad un opportuno “tool” i codici sorgenti o, talune volte anche quelli “compilati”, esso è in grado di ricostruire a ritroso il modello fino, ovviamente, alla fase di disegno
- Il processo diretto (engineering) e quello inverso (reverse engineering) determinano quello che in gergo viene definito round-trip engeneering.
- Nel mondo dell’ideale, la funzione di reverse non dovrebbe mai venir utilizzata… Però può capitare di dover “mettere le mani” su un codice scaricato da Internet o scritto in tempi precedenti e non opportunamente documentato

Struttura dell'UML

L'UML, analizzato con una metodologia Top-Down, è costituito da:

- Viste;
- Diagrammi;
- Elementi del modello;

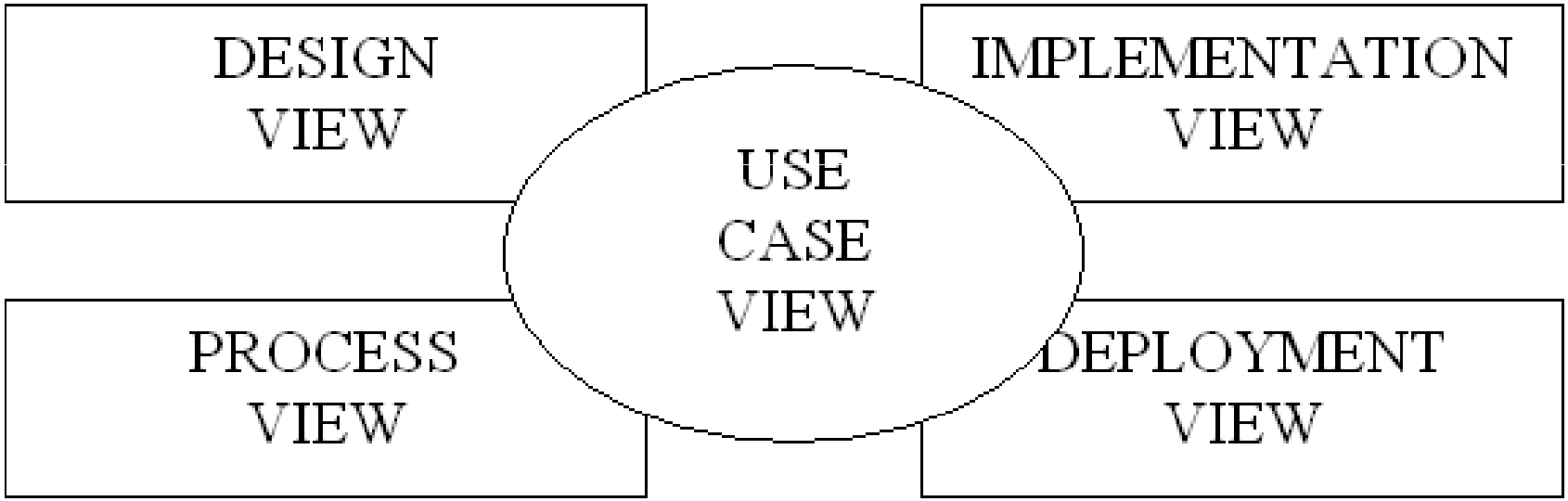
Struttura dell'UML

- Le **viste** mostrano i differenti aspetti di un sistema attraverso la realizzazione di un certo numero di diagrammi. Si tratta di astrazioni, ognuna delle quali analizza il sistema da modellare con un'ottica diversa
- I **diagrammi** permettono di esprimere le viste logiche per mezzo di grafici, ognuno dei quali è, tipicamente, destinato ad essere utilizzato per una particolare vista.
- Gli **elementi del modello** sono i concetti che permettono di realizzare i vari diagrammi, essi indicano gli attori, le classi, i packages, gli oggetti, ecc.

Le viste

Per ciò che concerne le viste, ne sono previste quattro:

- Use case;
- Logical;
- Component;
- Concurrency;
- Deployment.


DESIGN VIEW
IMPLEMENTATION VIEW
USE CASE VIEW
PROCESS VIEW
DEPLOYMENT VIEW

Use case view:

- Serve per analizzare i requisiti utente (COSA)
- Astrazione ad alto livello (black box)

Design/Logical view

- Serve per analizzare le funzionalità (COME)
- Specifiche interne al sistema (es. class diagram)

Implementation /Component view

- Organizzare codice (classi) in moduli (package)

Process/concurrency view

- Analisi non funzionale (processi e processori)

Deployment view → architettura fisica del sistema

I diagrammi

- Class diagram;
- Object diagram;
- Use case diagram;
- Sequence diagram;
- Collaboration diagram;
- Statechart diagram;
- Activity diagram;
- Component diagram;
- Deployment diagram.

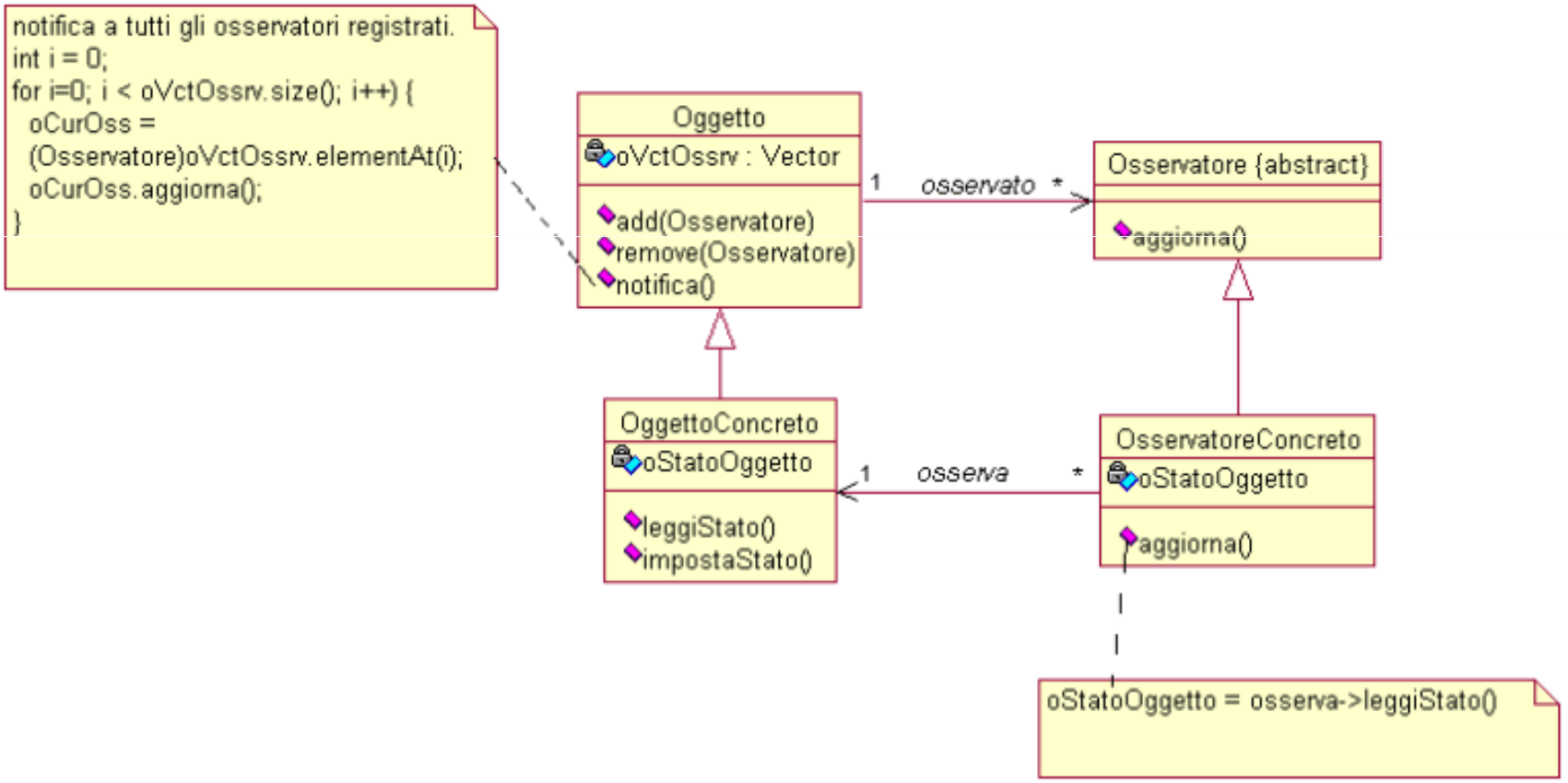
Class diagram

Descrive il tipo degli oggetti che compongono il sistema

- con tutti gli attributi e le operazioni
- eventualmente la loro visibilità
- e le relazioni statiche tra di loro
  - Associazioni
  - Generalizzazioni

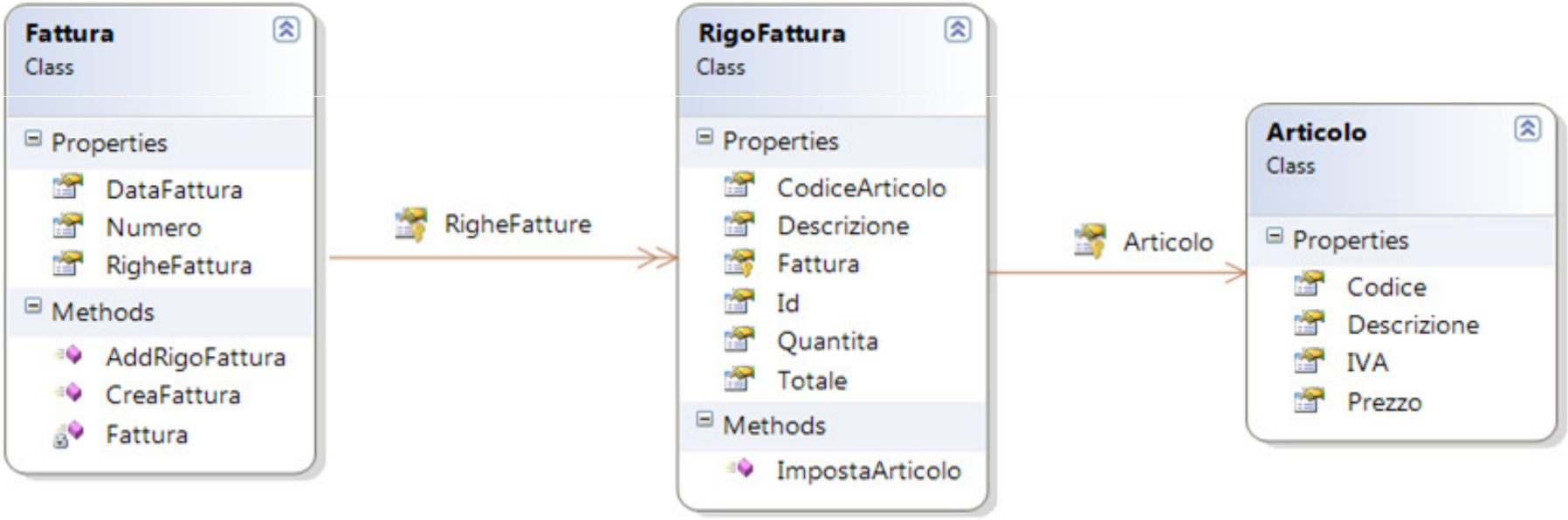
Class Diagram: esempio

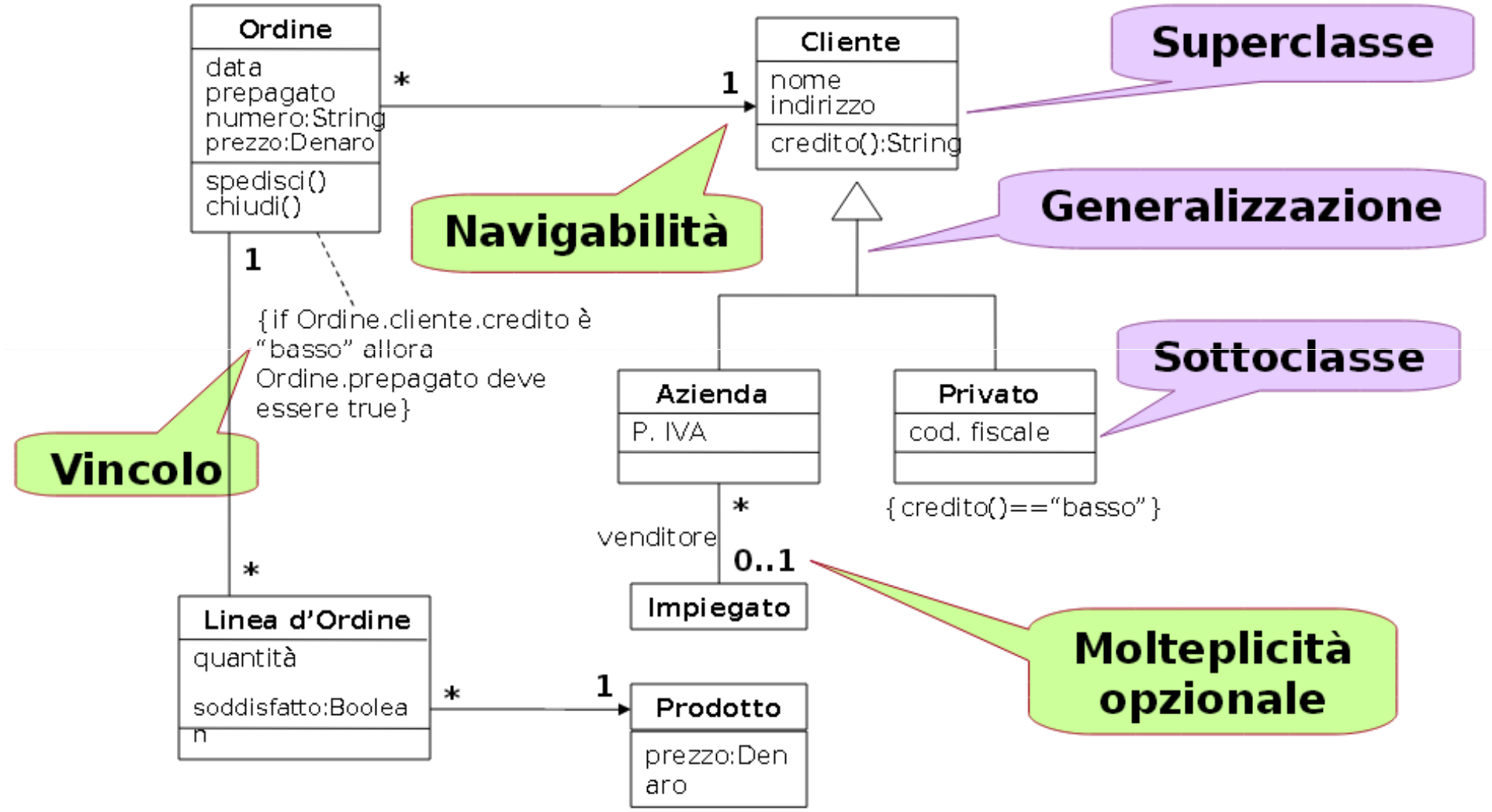
- Pattern dell'osservatore (Observer Pattern)

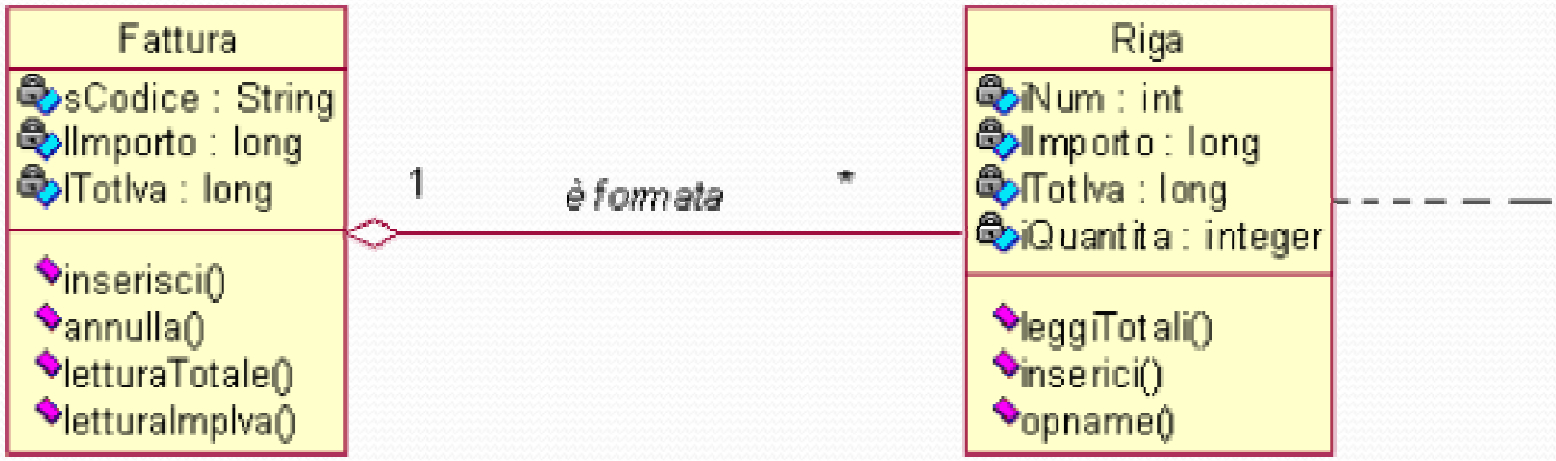


Esempio: Fattura e Articoli

- Estendibile con altre classi, es: cliente, fornitore, etc..

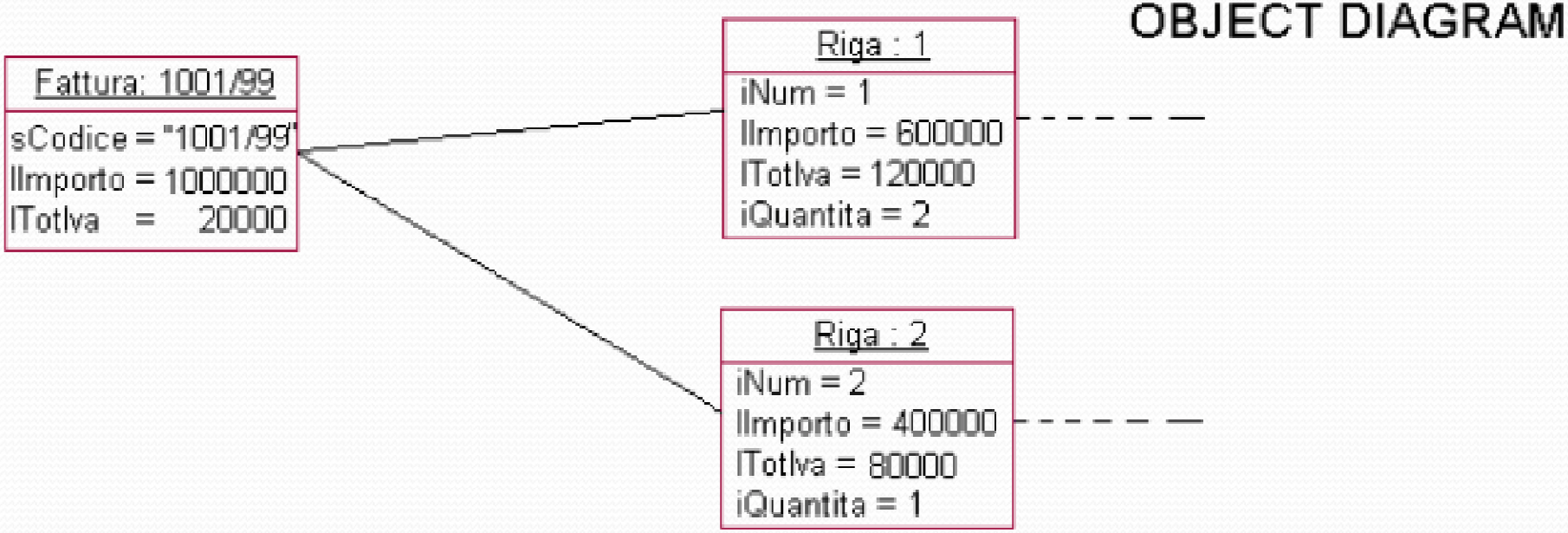



Ordine
data
prepagato
numero:String
prezzo:Denaro
spedisci()
chiudi()
*
1
Cliente
nome
indirizzo
credito():String
Superclasse
Navigabilità
Generalizzazione
1
{if Ordine.cliente.credito è "basso" allora Ordine.prepagato deve essere true}
Vincolo
Azienda
P. IVA
Privato
cod. fiscale
Sottoclasse
{credito()=="basso"}
*
venditore
0..1
Impiegato
Molteplicità opzionale
*
Linea d'Ordine
quantità
soddisfatto:Boolean
*
1
Prodotto
prezzo:Denaro



CLASS DIAGRAM

OBJECT DIAGRAM



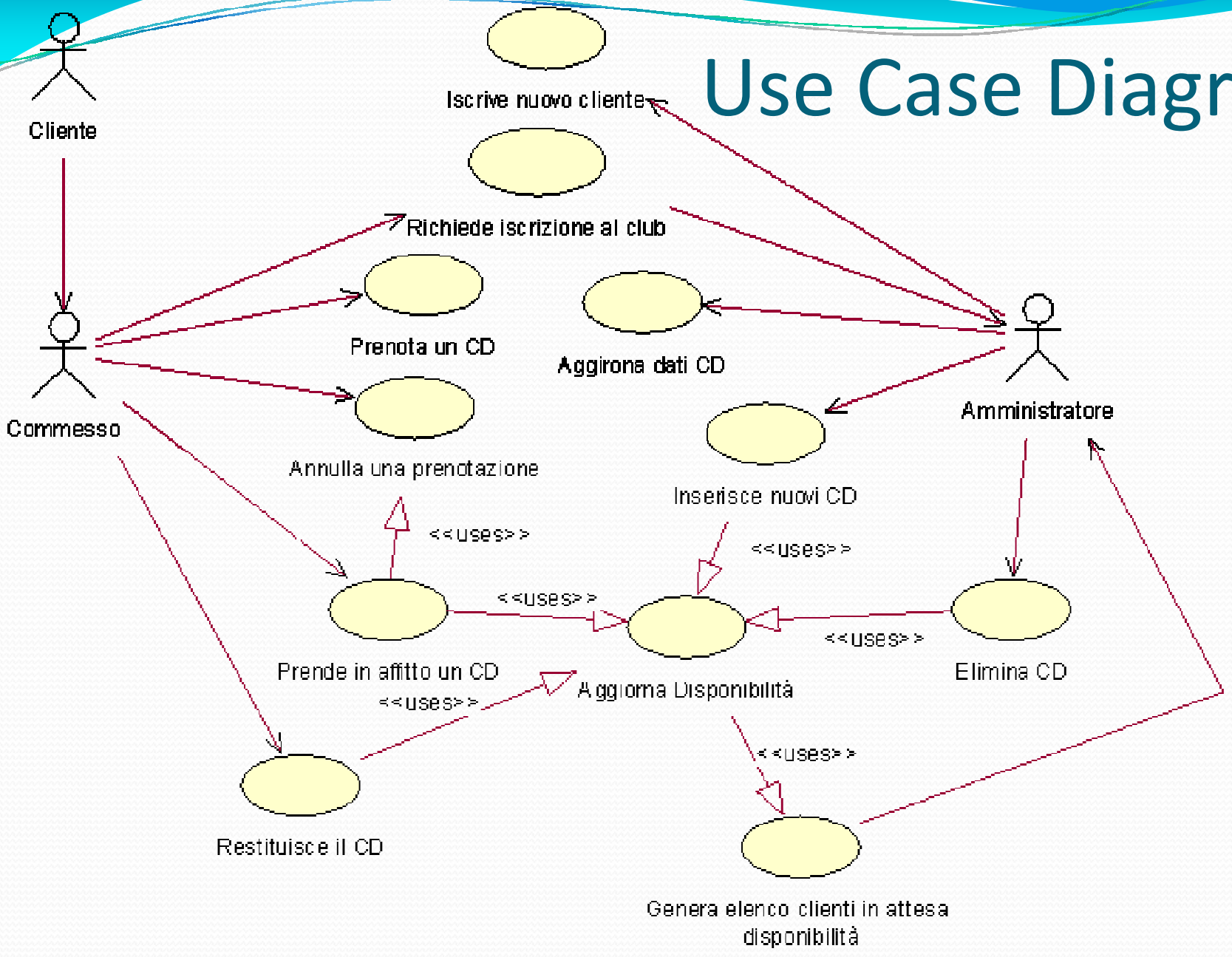
Class D. VS Object D.

- Gli objects diagrams mostrano un numero di oggetti istanze delle classi e i relativi legami espliciti
- Entrambi i diagrammi si occupano della parte statica del sistema e mostra un ipotetico esempio di un diagramma delle classi
- Mentre quest'ultimo è sempre valido, un diagramma degli oggetti rappresenta un'ipotetica istantanea del sistema. In sostanza lo si utilizza per esemplificare diagrammi delle classi, o suoi frammenti, ritenuti poco chiari o particolarmente complicati.

Strumenti OPEN per UML

- **Visual Paradigm for UML 10.2 Community Edition**
- http://www.visual-paradigm.com/
- Versione gratuita con licenza non commerciale

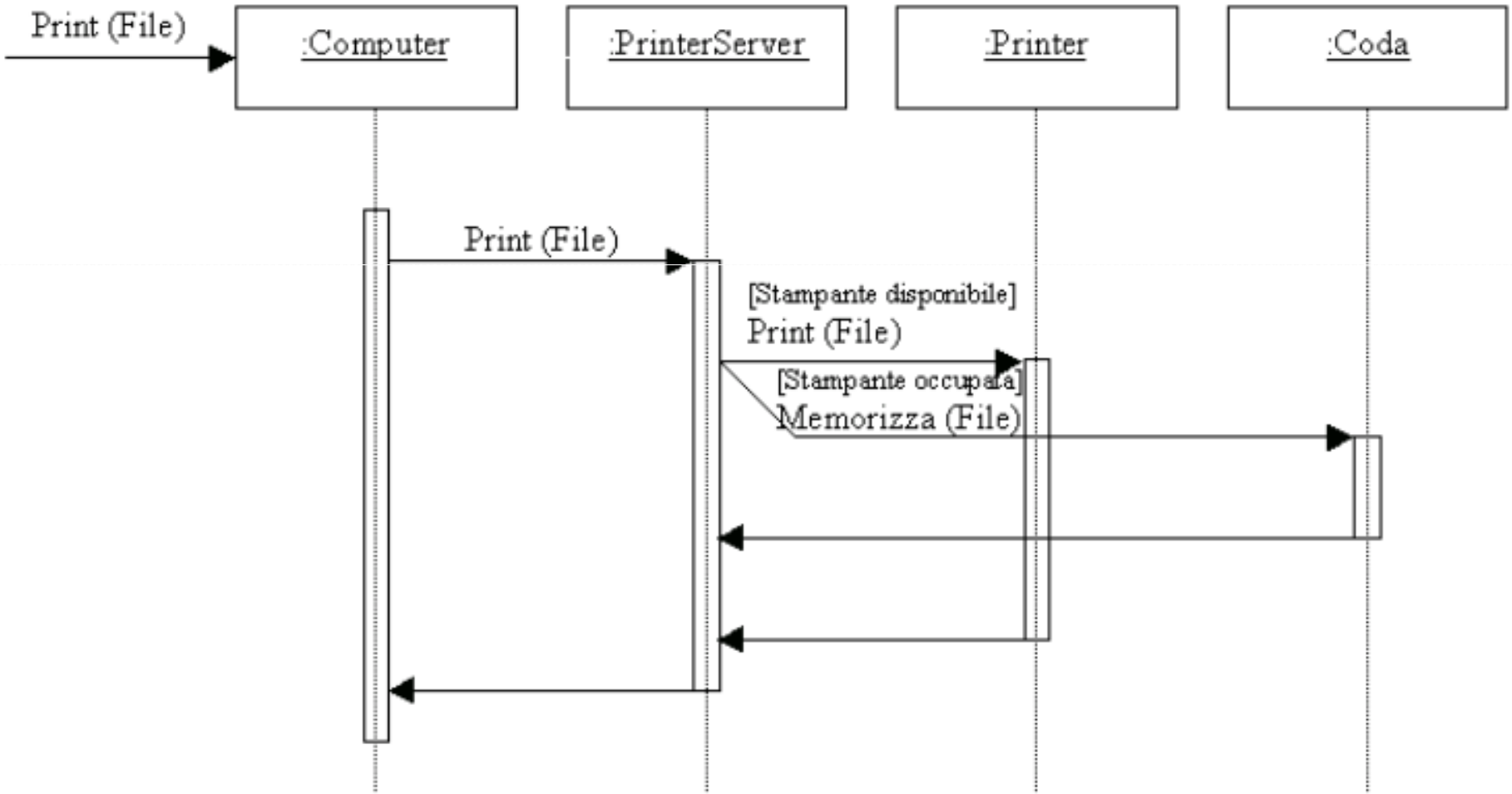
Use Case Diagram



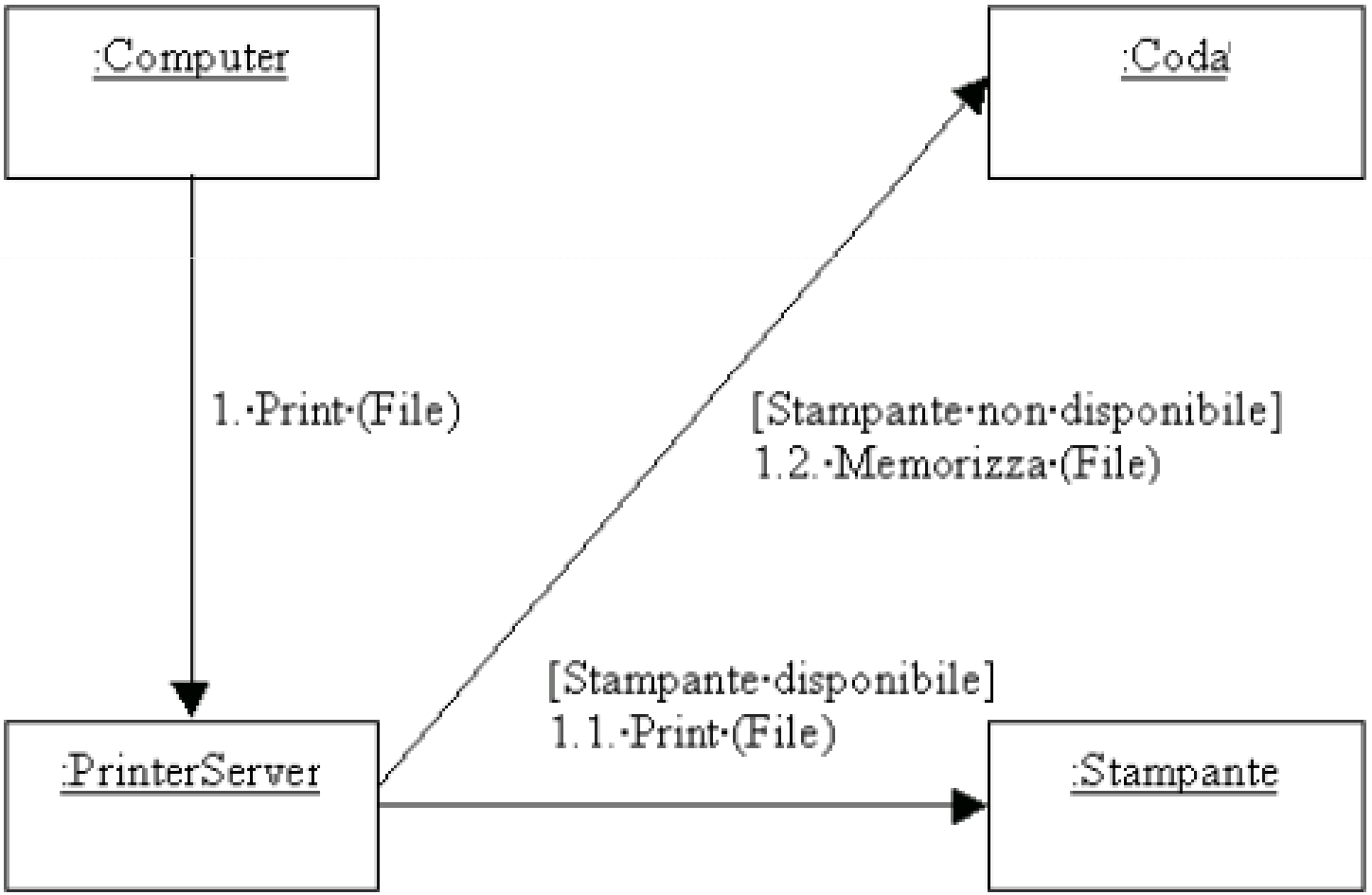
Sequence D VS Collaboration D

- I diagrammi di sequenza e collaborazione mostrano appunto le interazioni tra oggetti del sistema.
- Si occupano di modellare il comportamento dinamico del sistema.
- I diagrammi di sequenza evidenziano l'ordine temporale dello scambio di messaggi
- Idiagrammi di collaborazione mettono in risalto l'organizzazione degli oggetti che si scambiano i messaggi.

Sequence diagram



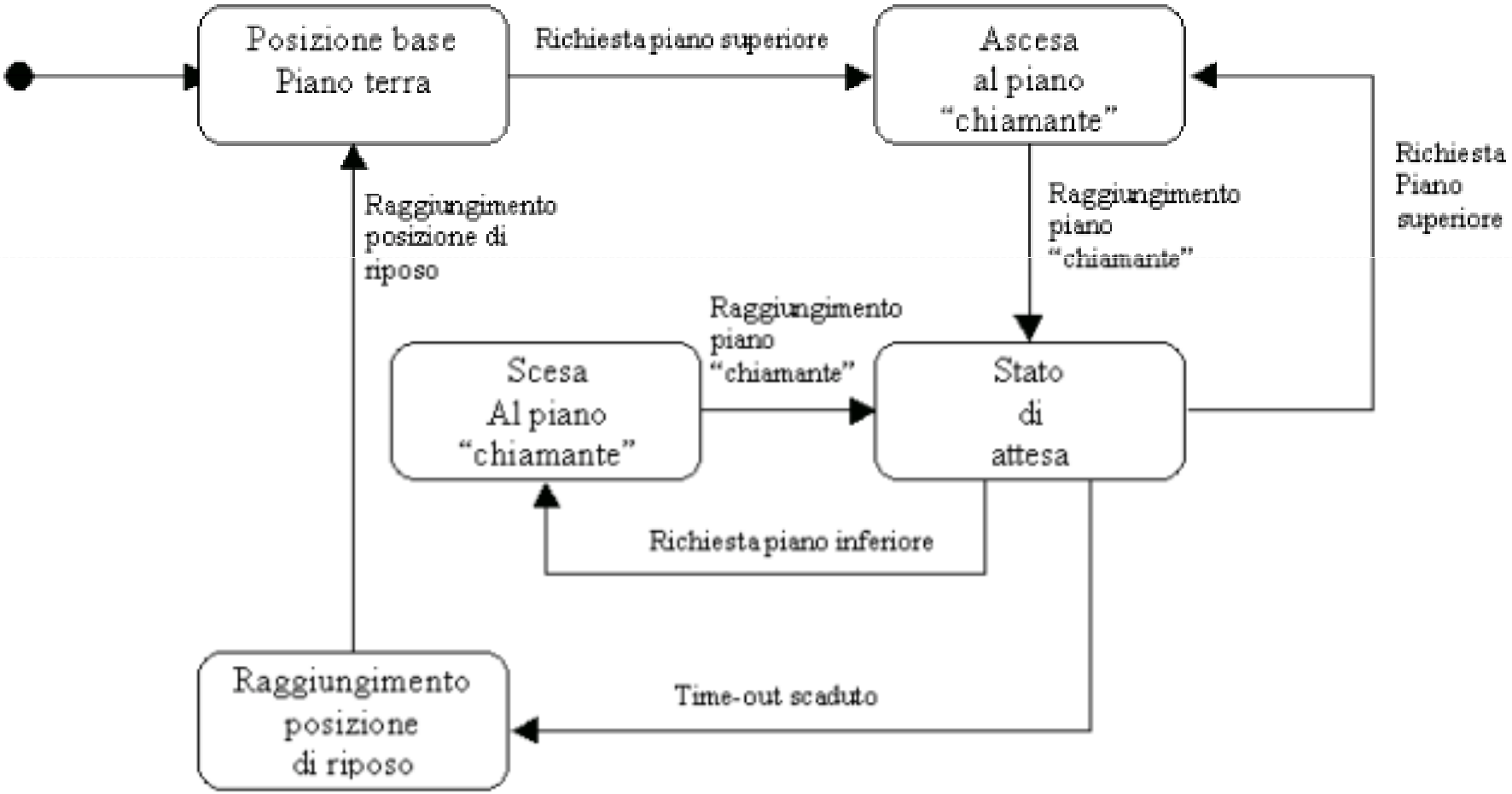
Collaboration Diagram



Altri diagrammi…

- I diagrammi di stato mostrano un automa a stati finiti, e pertanto sono costituiti da stati, transazioni, eventi e attività.
- I diagrammi di attività sono simili ai flow chart e contengono gli elementi decisionali e condizionali.
- I diagrammi dei componenti mostrano la struttura fisica del codice in termini di componenti e di reciproche dipendenze. (Package)
- Idiagrammi di dispiegamento che mostrano l'architettura hardware e software del sistema.

State diagram



Deployment Diagram

- Illustrano gli elementi fisici del sistema (computer, reti, device fisici,…) e i moduli software da installare su ciascuno di essi.