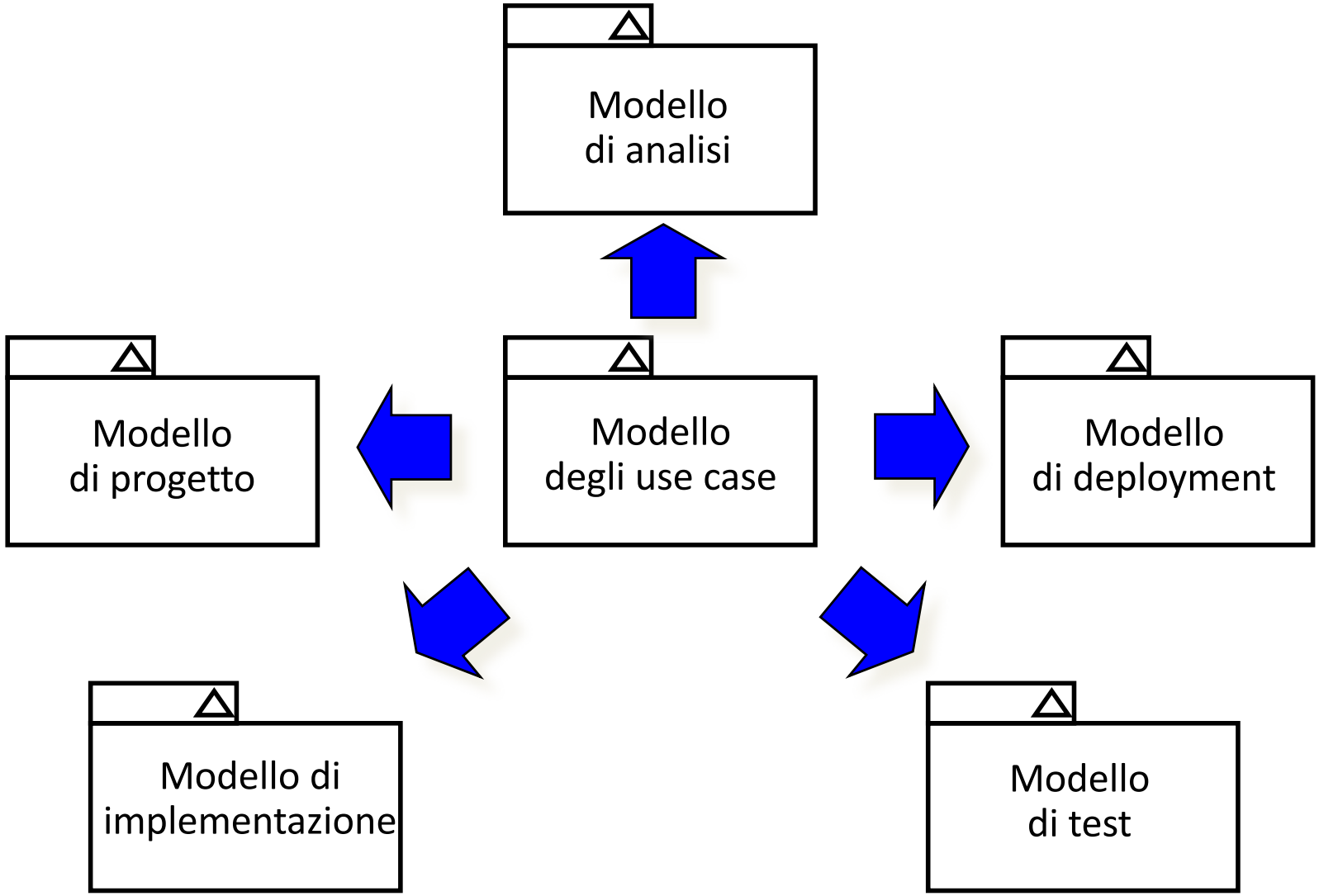
UML

Unified Modeling Language

Modellazione

- **Modello** è una descrizione astratta del sistema
  - …in cui è inclusa anche una descrizione dell'ambiente in cui il sistema dovrà operare
- Modellare il sistema aiuta gli analisti a capire le funzionalità del sistema
- I modelli del sistema sono usati per comunicare con i committenti
- Diversi modelli presentano il sistema da diverse prospettive

Alcuni modelli interessanti



UML

- Il software non dispone ancora di tecniche efficaci per descriverne la struttura, le funzionalità, e le prestazioni
- UML cerca di rimediare a questa situazione
  - Standard OMG (Object Management Group)
  - Oggi siamo alle versione 2.X
  - Astrazioni indipendenti dal linguaggio di programmazione

Alcuni diagrammi

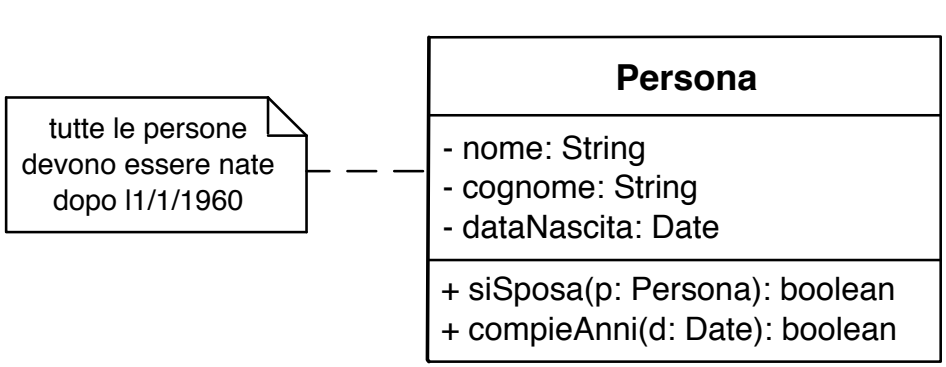
- Diagrammi di **struttura**
  - …diagrammi delle classi, diagrammi degli oggetti, diagrammi dei componenti, diagrammi delle strutture composte, diagrammi dei package e diagrammi di deployment
- Diagrammi di **comportamento**
  - …diagrammi dei casi d'uso, diagrammi delle attività e diagrammi delle macchine a stati
- Diagrammi di **interazione**
  - …diagrammi di sequenza, diagrammi di comunicazione, diagrammi di temporizzazione e diagrammi di interazione generale

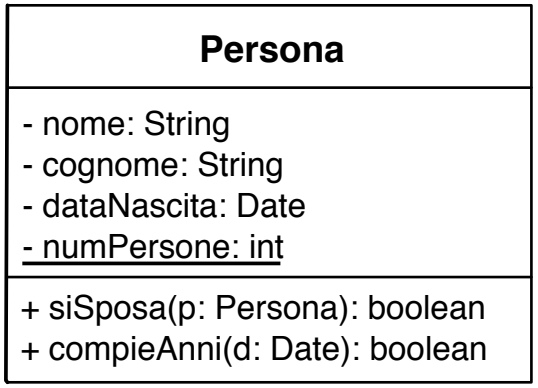
Dettagli nella descrizione

- UML consente di esprimere graficamente livelli crescenti di dettaglio nella descrizione delle entità
- Questi livelli crescenti di dettaglio sono spesso inappropriati o addirittura completamente fuori luogo nella specifica dei requisiti
- Diventano invece essenziali nella descrizione dell'architettura della soluzione, dove le classi corrispondono esattamente alle classi della soluzione in Java

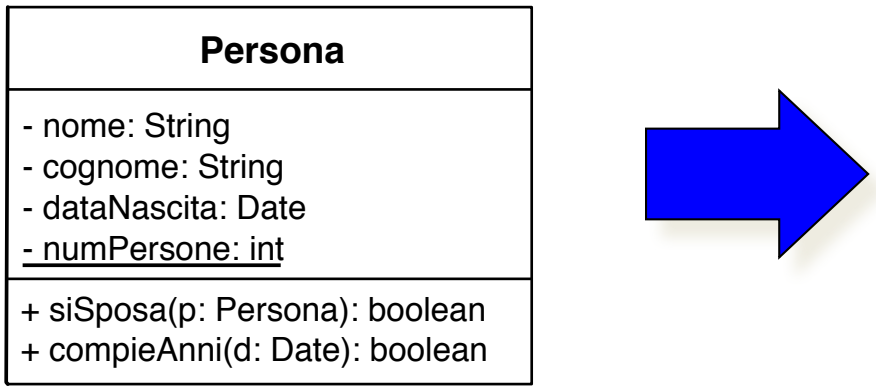
Diagramma delle classi

- Descrizione statica
- Una classe è composta da tre parti
  - Nome
  - Attributi (lo stato)
  - Metodi (il comportamento)
- Attributo: visibilità nome: tipo [molteplicità] = default {stringa di proprietà}
- Metodo: visibilità nome (lista parametri): tipo di ritorno {stringa di proprietà}
- Visibilità: + public, - private, # protected, ~ friendly
- Parametro: direzione nome: tipo = default



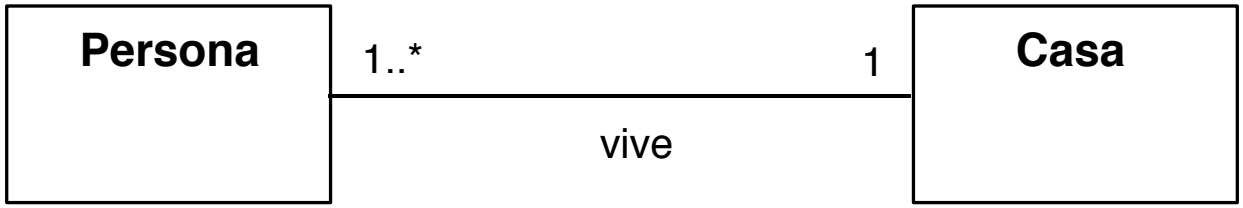


Traduzione



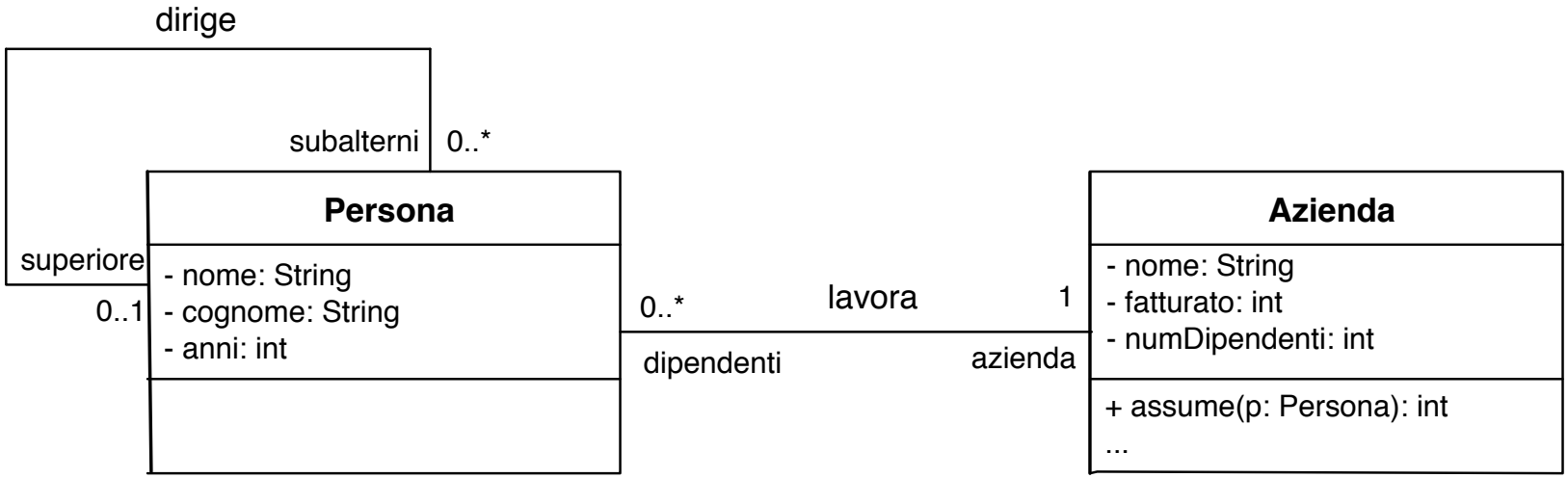

```
public class Persona {
  private String nome;
  private String cognome;
  private Date dataNascita;
  private static int numPersone;

  public boolean siSposa(Persona p) {
     …
  }

  public boolean compieAnni(Date d) {
     …
  }
}
```

Associazioni

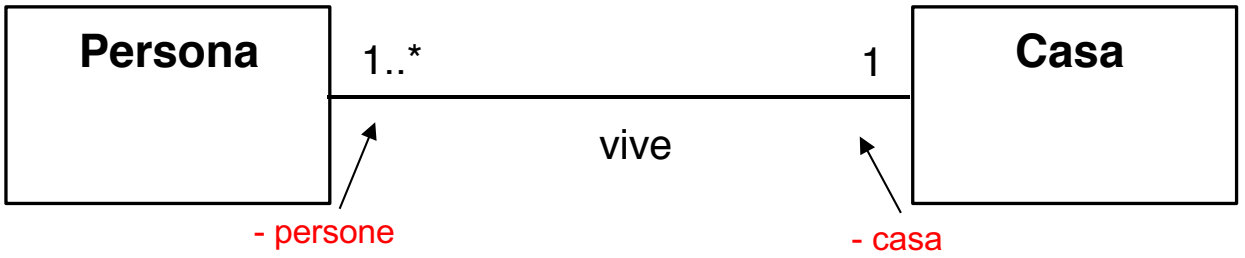


- Un’associazione indica una relazione tra classi
  - …ad esempio persona che lavora per una azienda
- Un’associazione può avere
  - Un nome (solitamente un verbo)
  - I ruoli svolti dalle classi nell’associazione
- Gli estremi di un’associazione
  - Sono “attributi impliciti”
  - Hanno visibilità come gli attributi normali
  - Hanno una molteplicità
    - 1, 0..1, 1..*, 4, 6-12

Esempio


dirige
subalterni
0..*
Persona
superiore
0..1
- nome: String
- cognome: String
- anni: int
0..*
lavora
1
dipendenti
azienda
Azienda
- nome: String
- fatturato: int
- numDipendenti: int
+ assume(p: Persona): int
...

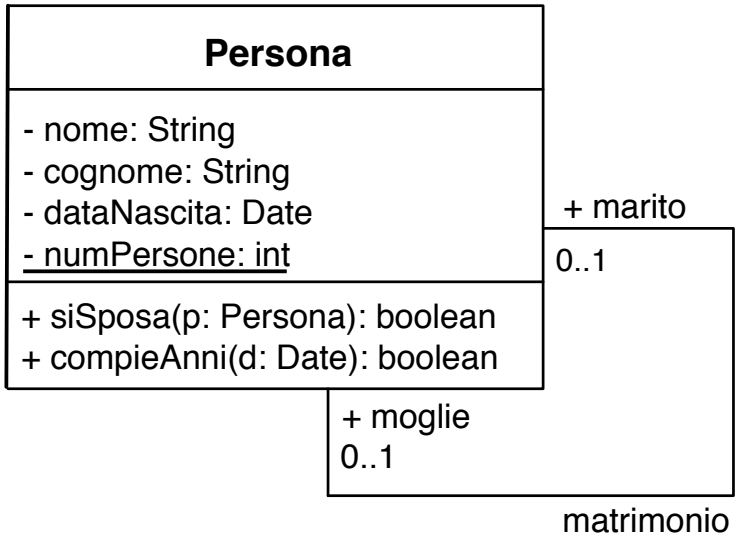
Esempio




```
class Persona {
    …
    private Casa casa;
    …
}
```

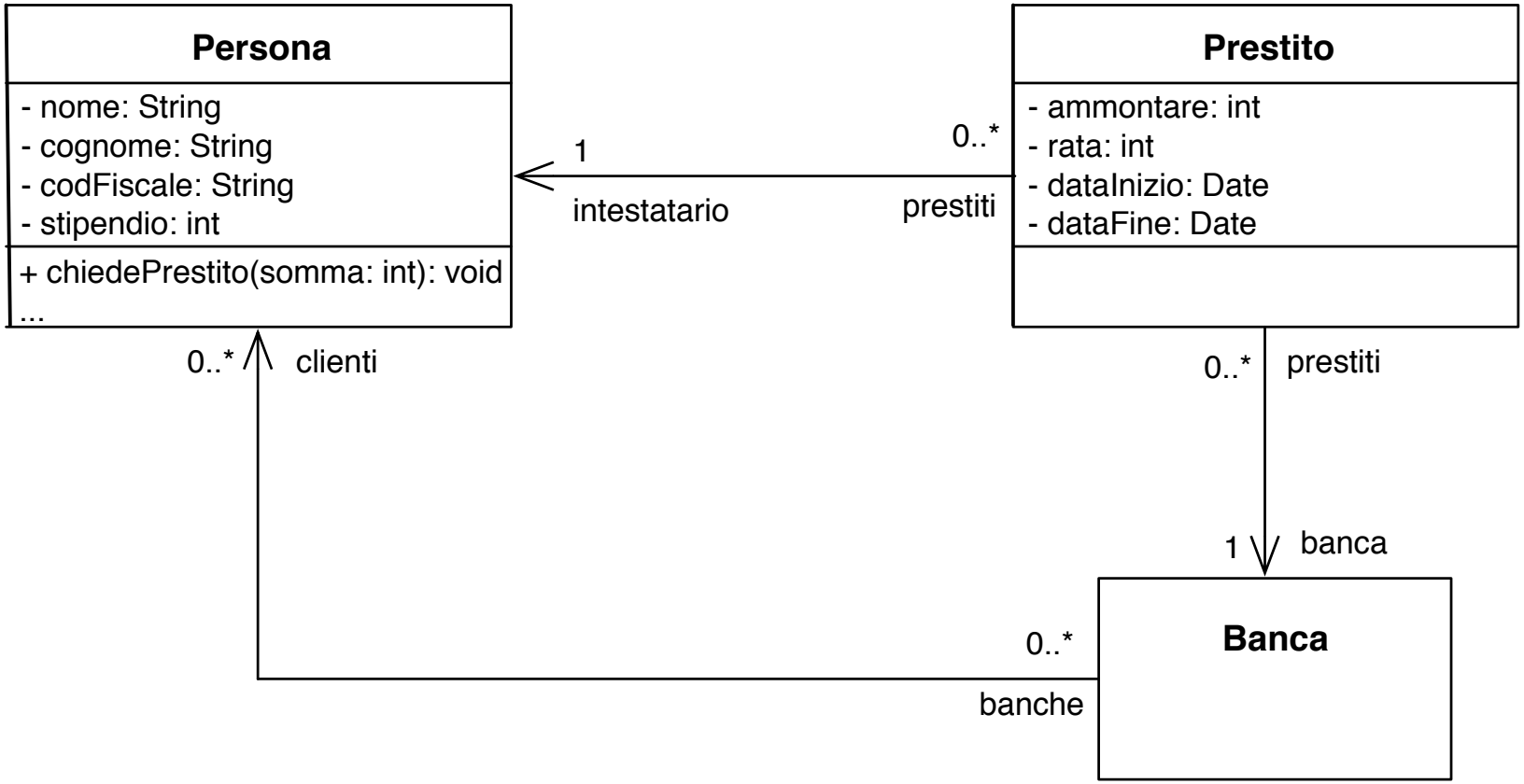
```
class Casa {
    …
    private Persona[] persone;
    …
}
```

Esempio




```
class Persona {
   private String nome;
   private String cognome;
   private Date dataNascita;
   private static int numPersone;
   public Persona marito;
   public Persona moglie;

   public boolean siSposa(Persona p) {
      …
   }

   public boolean compieAnni(Date d) {
      …
   }
}
```

Esempio

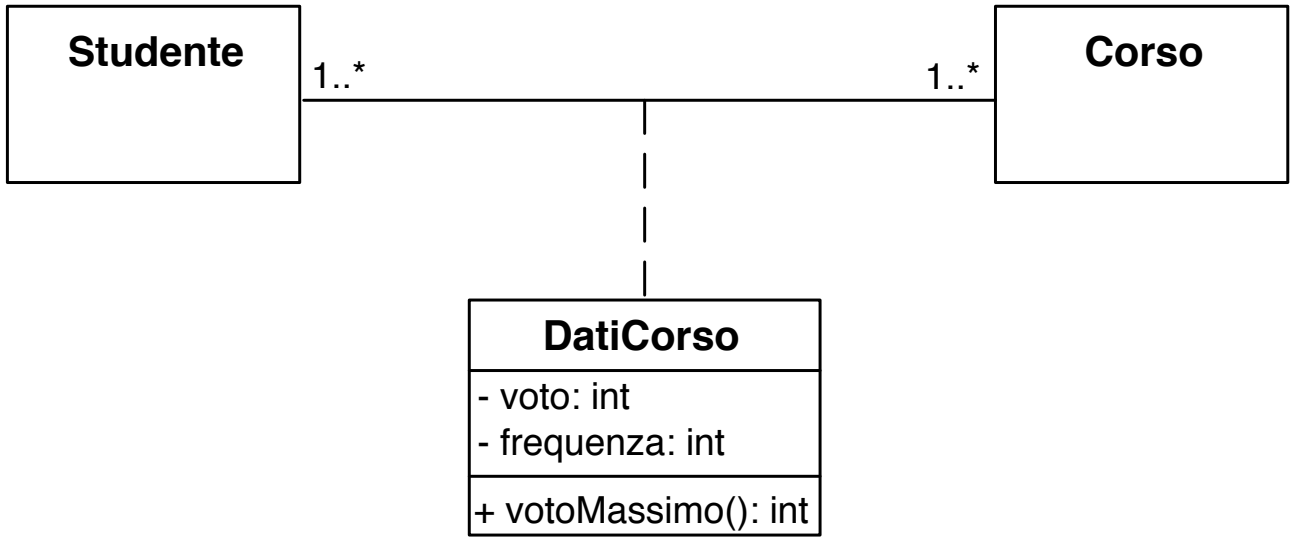

Persona
- nome: String
- cognome: String
- codFiscale: String
- stipendio: int
+ chiedePrestito(somma: int): void
...
Prestito
- ammontare: int
- rata: int
- dataInizio: Date
- dataFine: Date
1
intestatario
0..*
prestiti
0..*
clienti
0..*
prestiti
1
banca
Banca
0..*
banche

Esempio

```
class Persona {
   private String nome;
   private String cognome;
   private String codFiscale;
   private int stipendio;
}

class Banca {
   private Persona[] clienti;
}
```

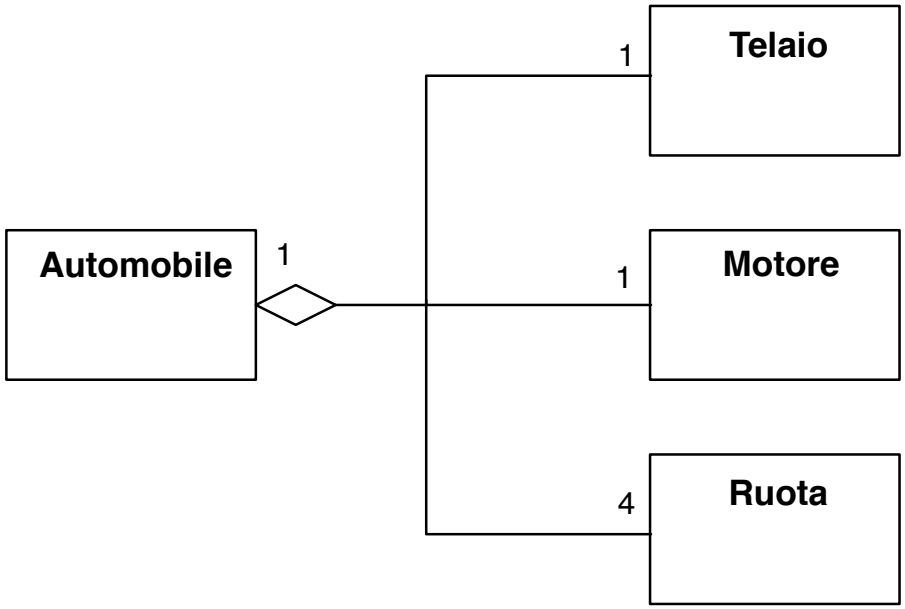
```
class Prestito {
   private int ammontare;
   private int rata;
   private Date dataInizio;
   private Date dataFine;
   private Persona intestatario;
   private Banca banca;
}
```

Classi associazione



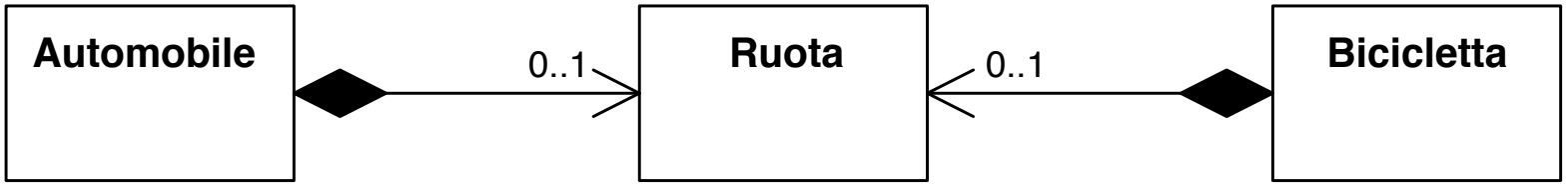
Aggregazioni

- Le aggregazioni sono una forma particolare di associazione
- Una parte è in relazione con un oggetto (**part-of**)



Composizioni

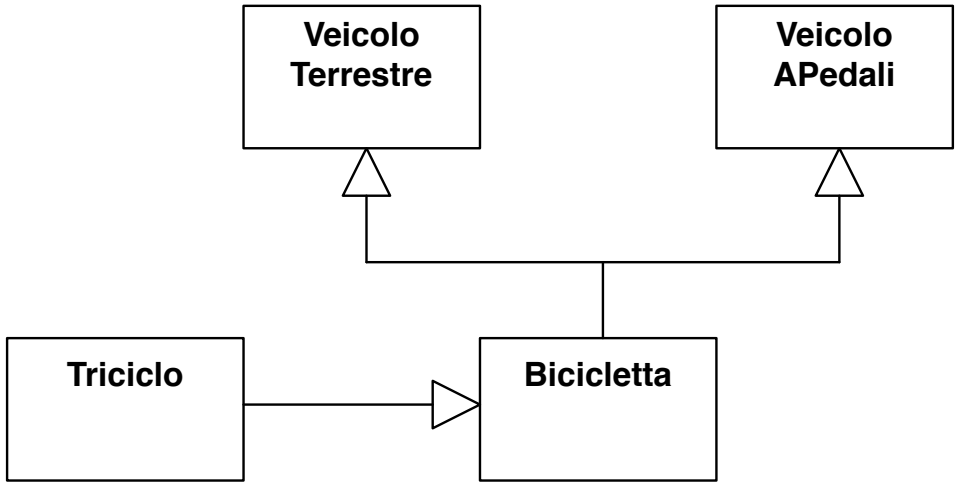
- Una relazione di composizione è un'aggregazione forte
  - Le parti componenti **non esistono** senza il contenitore
    - Creazione e distruzione avvengono nel contenitore
    - Le parti componenti non sono parti di altri oggetti



- In Java aggregazioni e composizioni si traducono allo stesso modo
  - In C++ esistono modi differenti

Ereditarietà (generalizzazione)

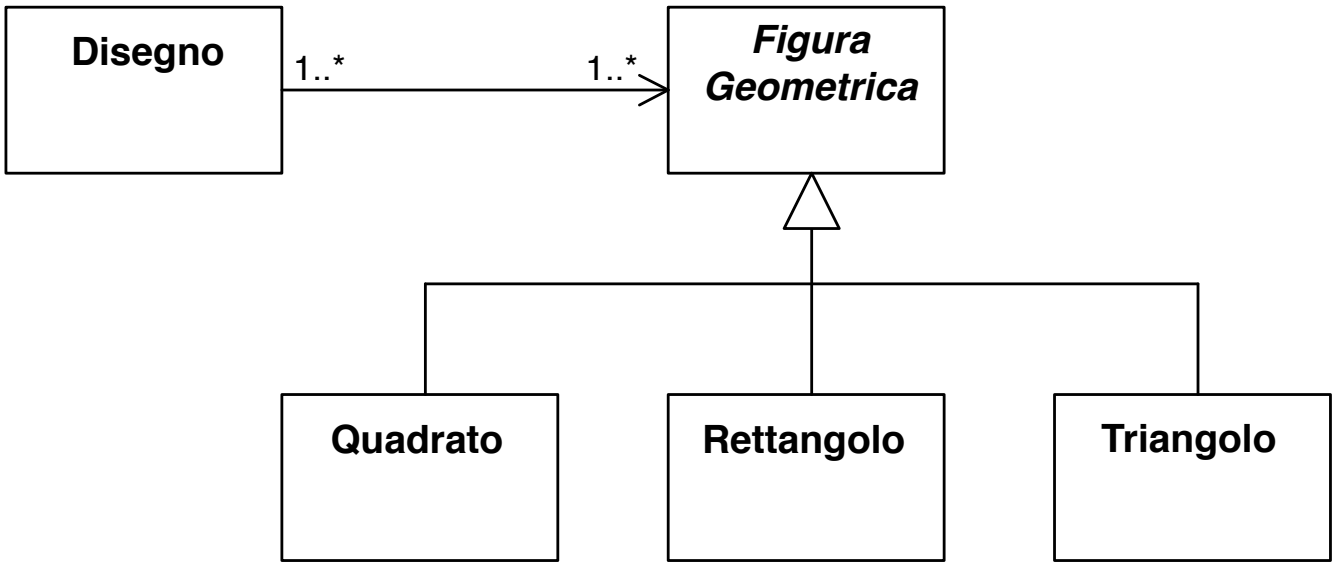
- Esplicita eventuali comportamenti comuni



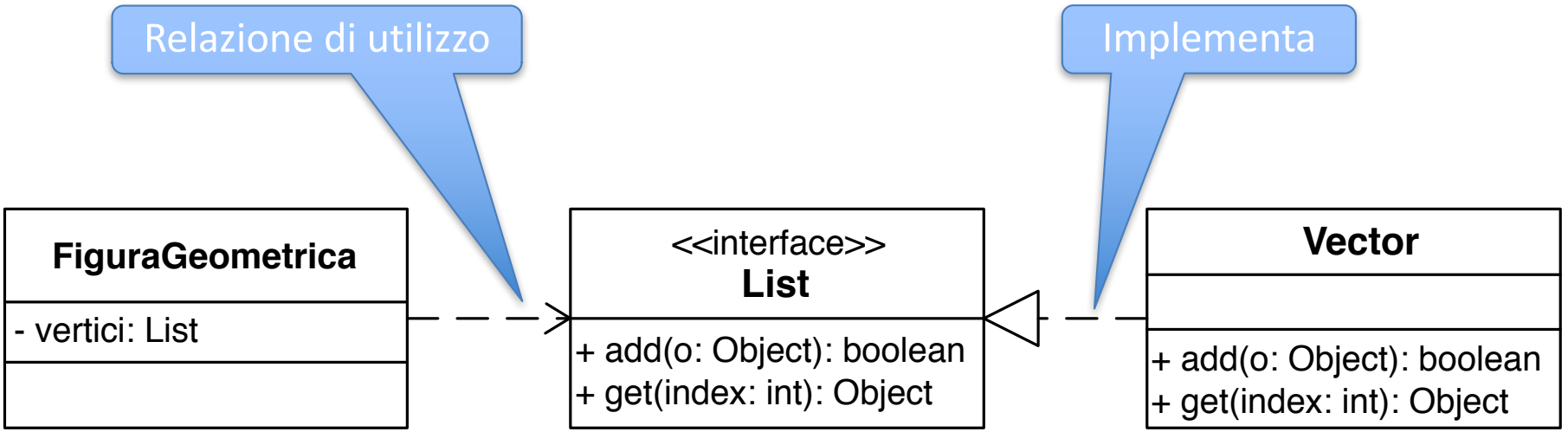

```
class Triciclo extends Bicicletta {
   …
}
```

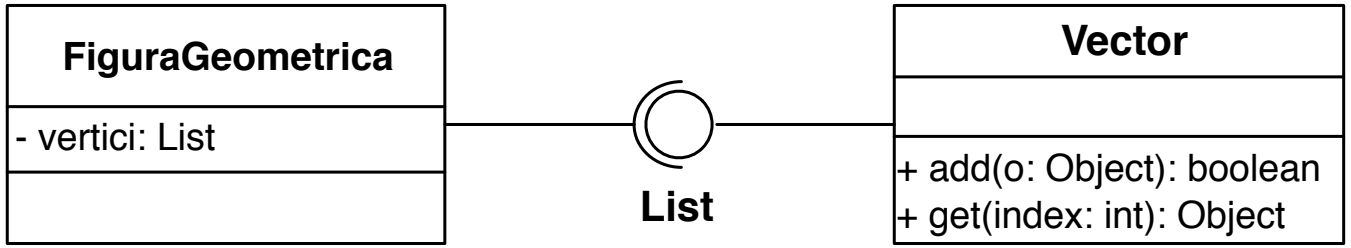
- Possibilità di ereditare da più classi
  - Vietato in Java
- Può portare a conflitti fra attributi o servizi con lo stesso nome ereditati da classi diverse

Classi astratte

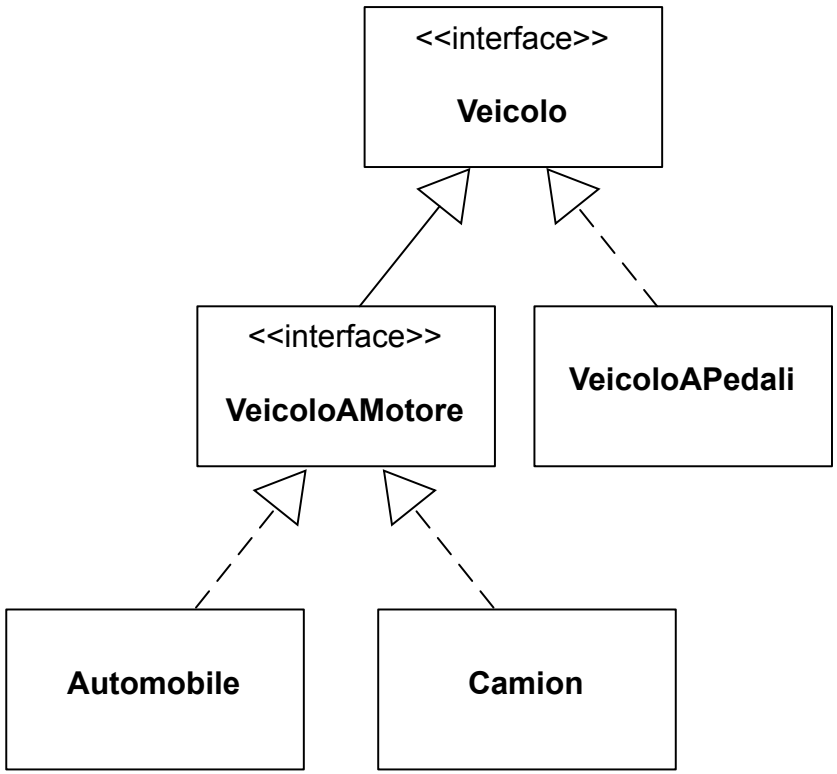


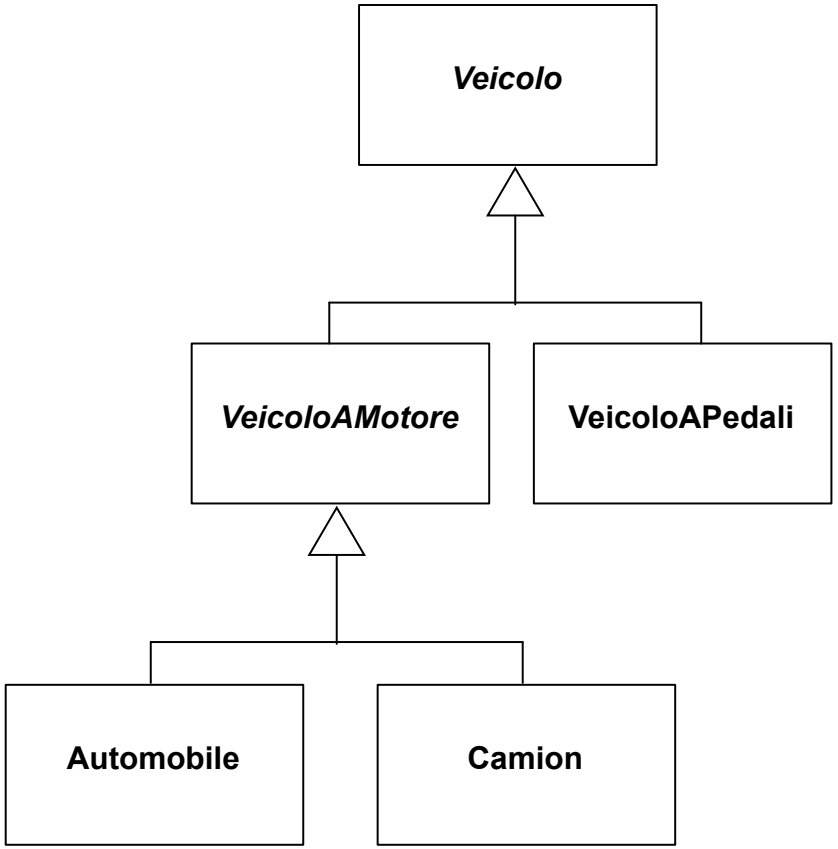
Interfacce


Relazione di utilizzo
Implementa
FiguraGeometrica
- vertici: List
<<interface>>
List
+ add(o: Object): boolean
+ get(index: int): Object
Vector
+ add(o: Object): boolean
+ get(index: int): Object


FiguraGeometrica
- vertici: List
List
Vector
+ add(o: Object): boolean
+ get(index: int): Object

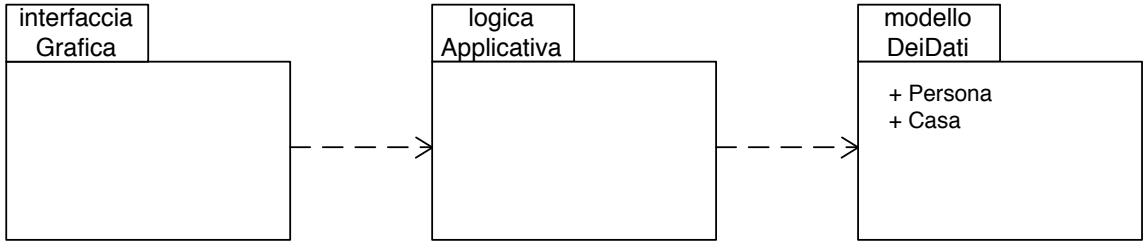
Interfacce ed ereditarietà

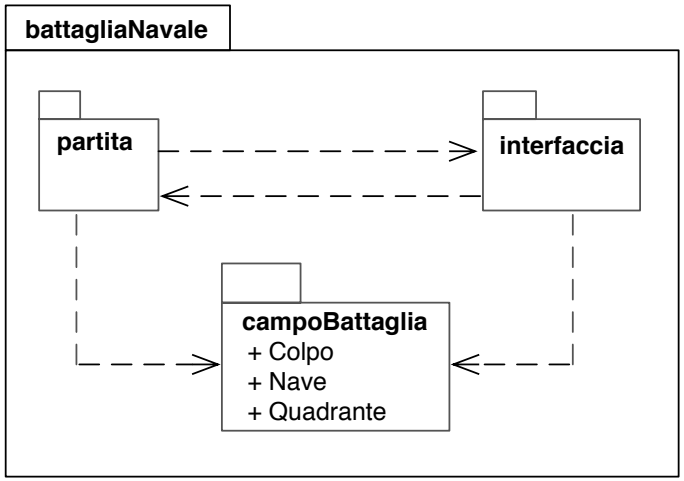


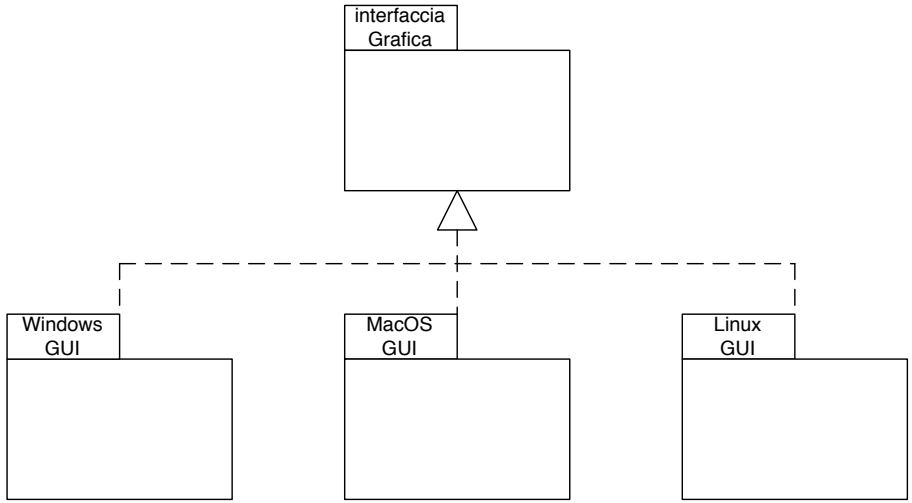


Package

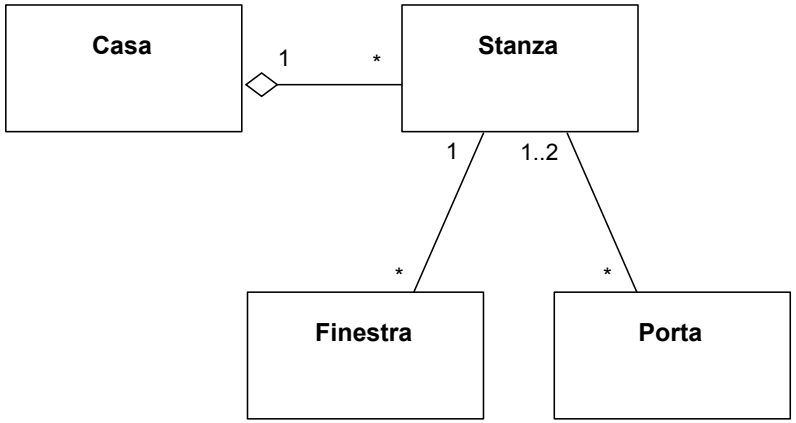
- Decomposizione gerarchica e dipendenze tra package
  - In Java esiste un concetto simile

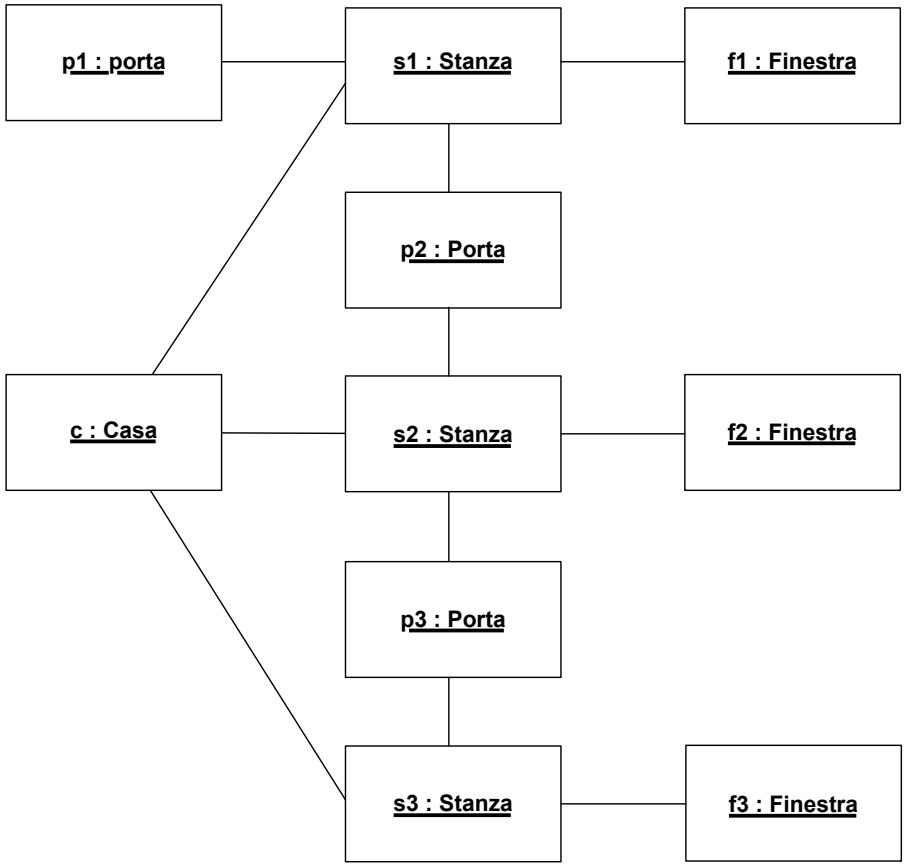






Dalle classe agli oggetti

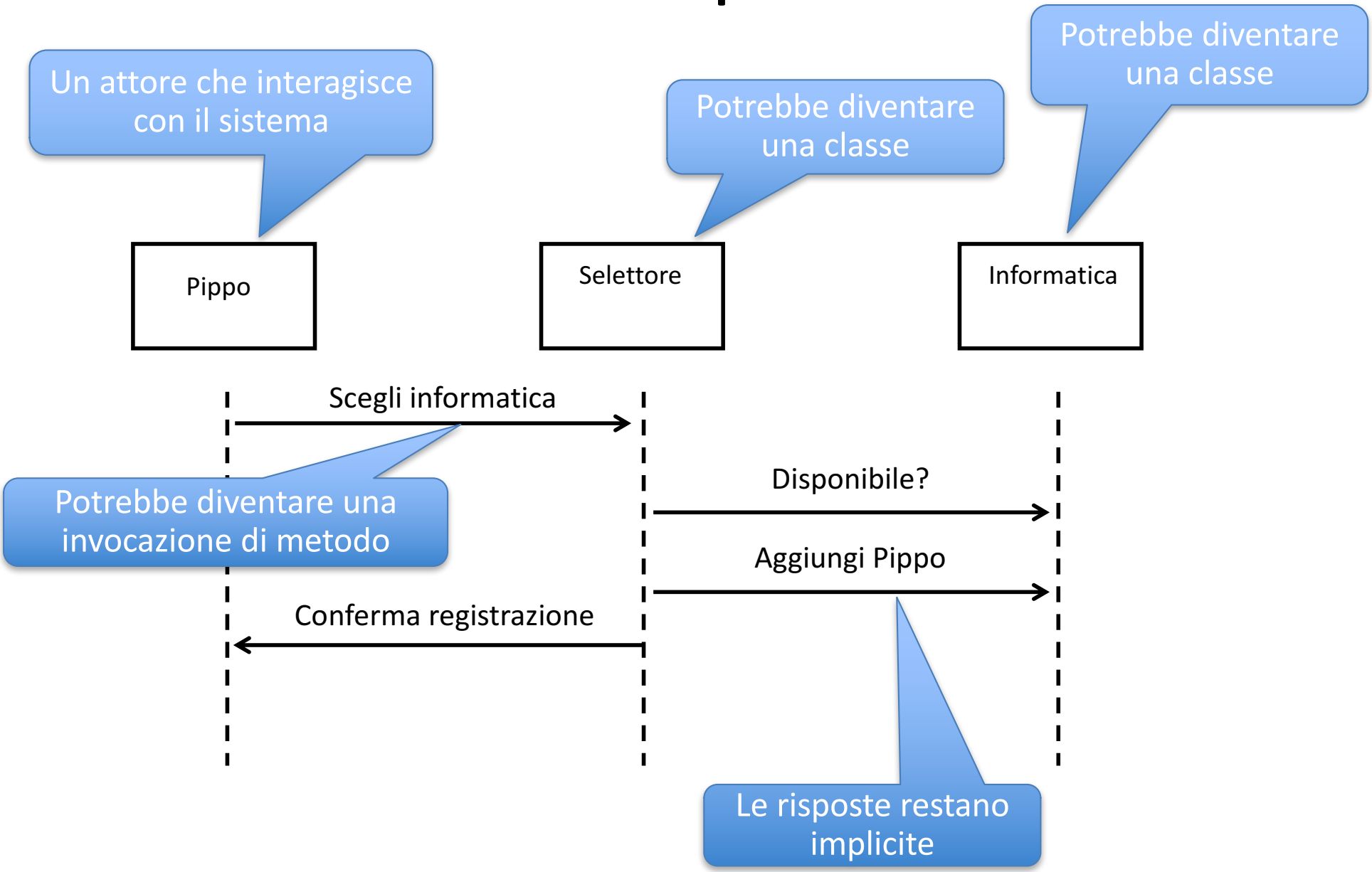




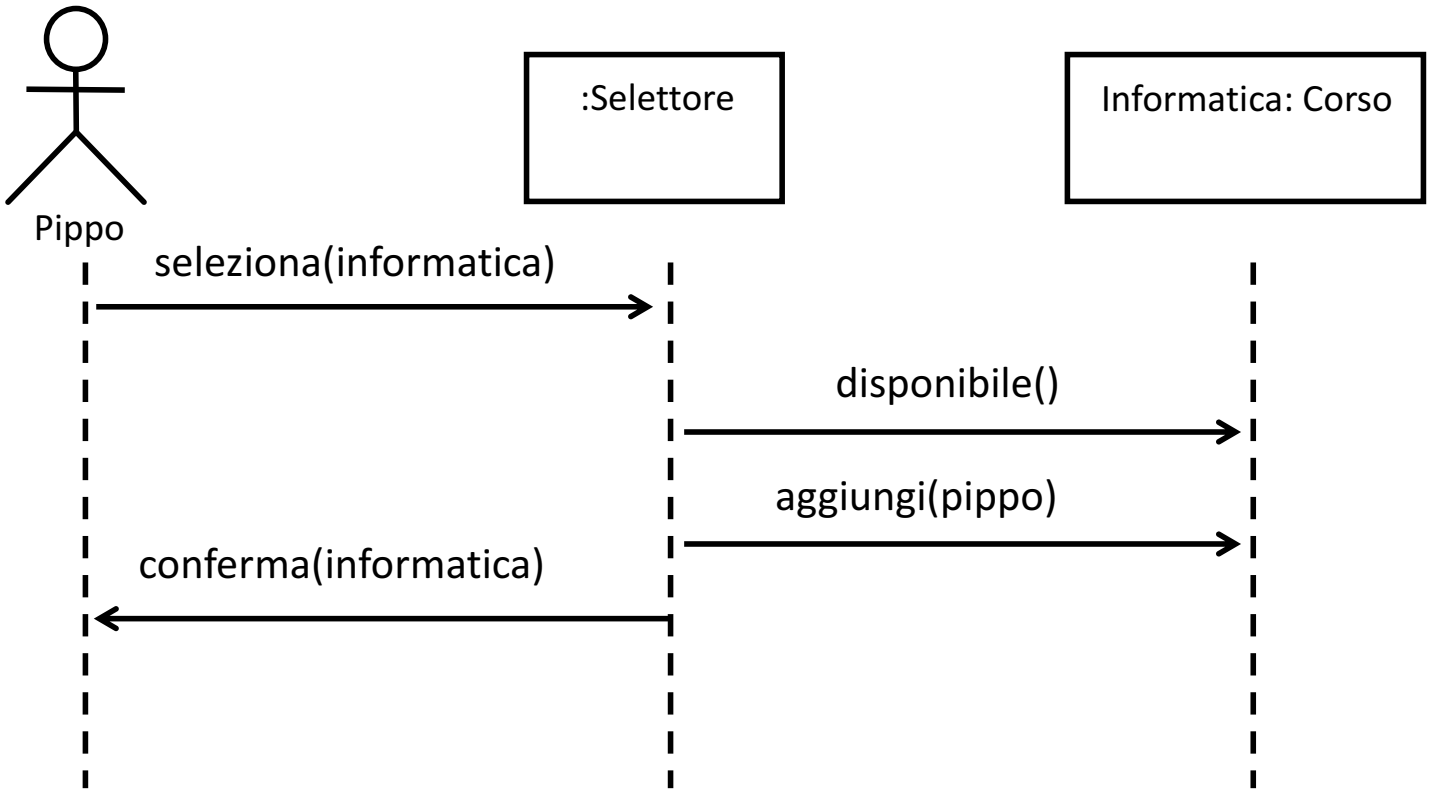
Diagrammi di interazione

- Descrivono il comportamento **dinamico** di un gruppo di oggetti che “interagiscono” per risolvere un problema
- Sono utilizzati per rappresentare scenari in termini
  - Entità (oggetti)
  - Messaggi scambiati (metodi)
- UML include
  - Diagrammi di sequenza
  - Diagrammi di comunicazione

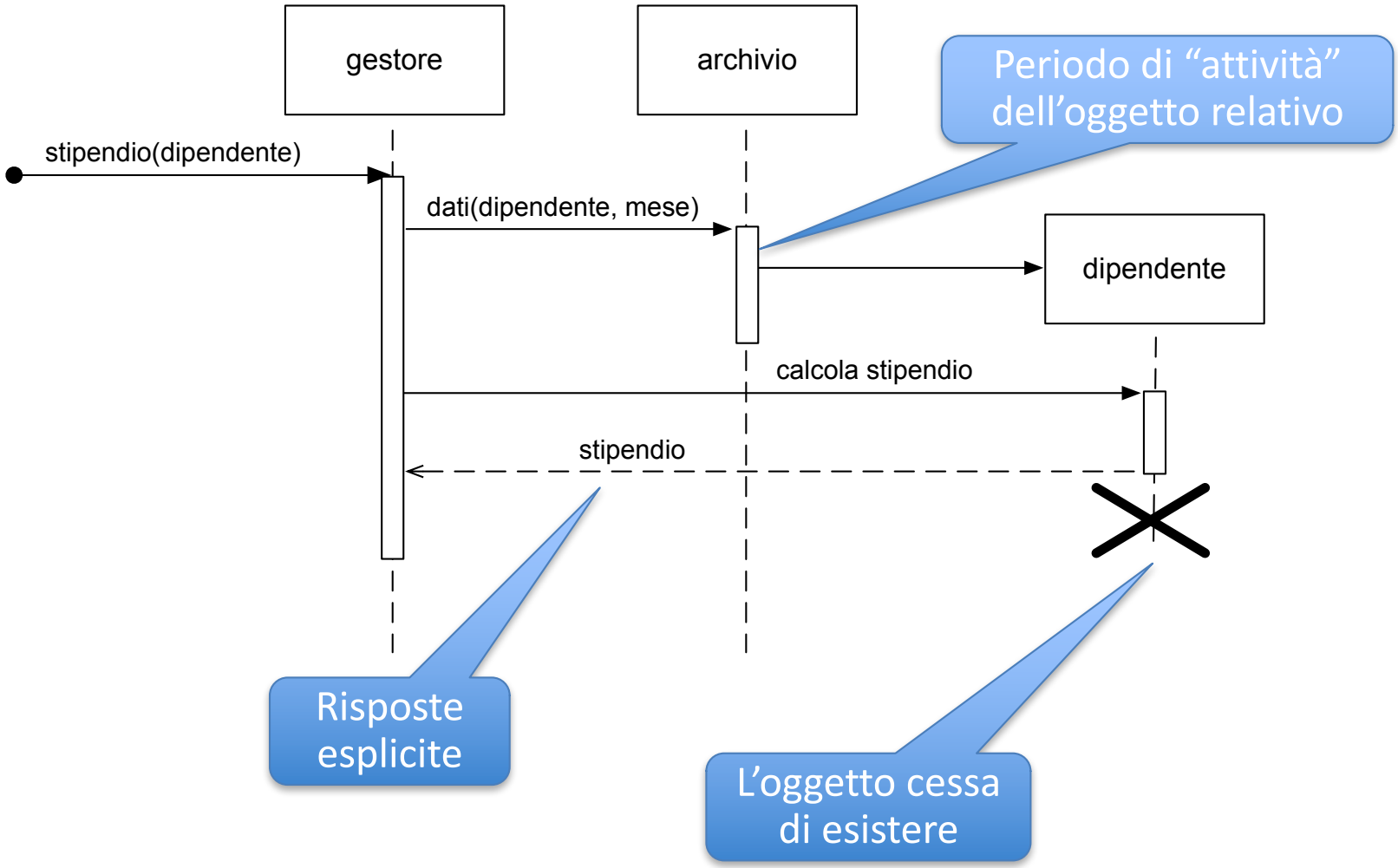
Esempio


Un attore che interagisce con il sistema
Potrebbe diventare una classe
Potrebbe diventare una classe
Pippo
Selettore
Informatica
Scegli informatica
Potrebbe diventare una invocazione di metodo
Disponibile?
Aggiungi Pippo
Conferma registrazione
Le risposte restano implicite

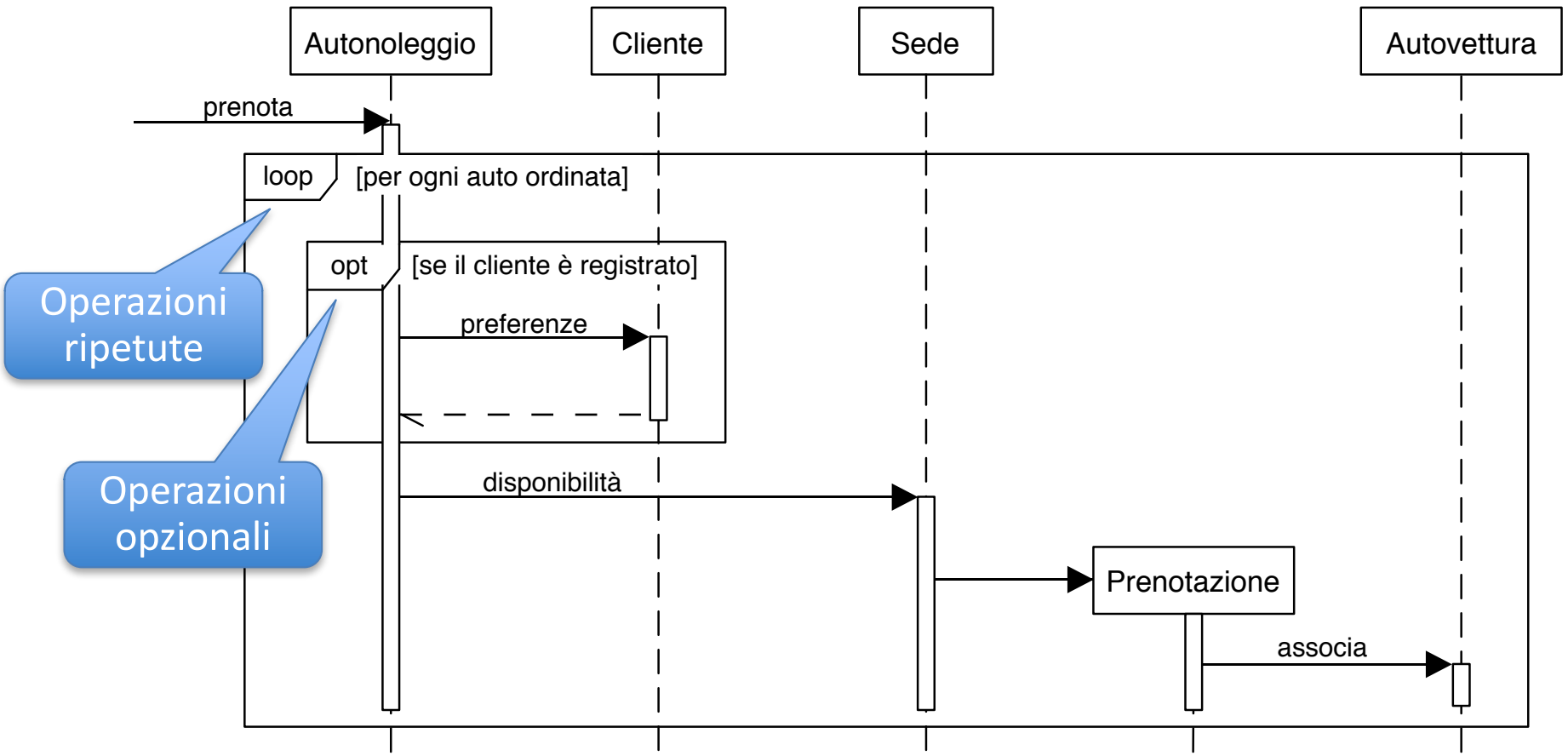
Raffinamento



Qualcosa in più



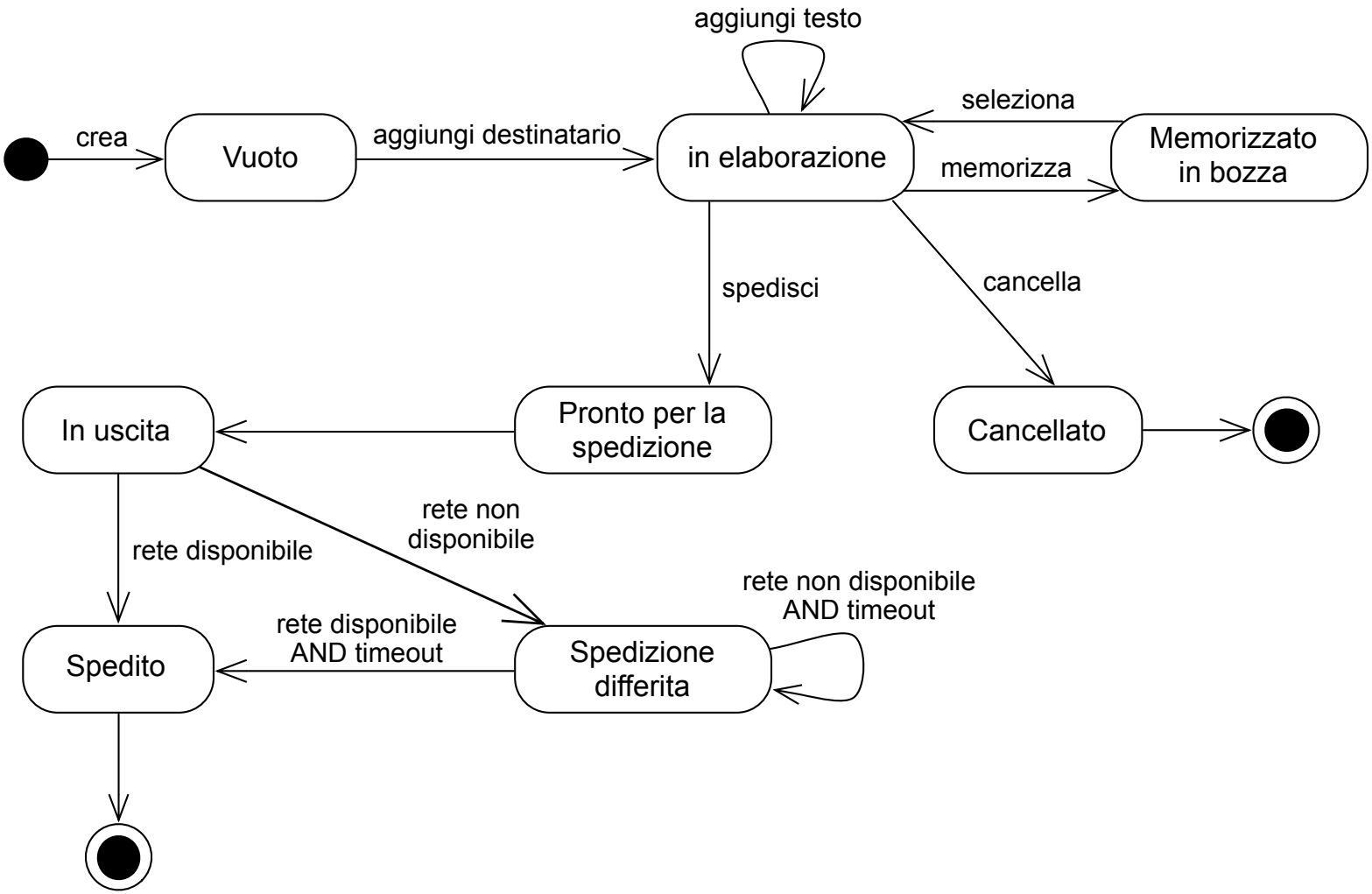
Frames di interazione



Macchine a stati finiti

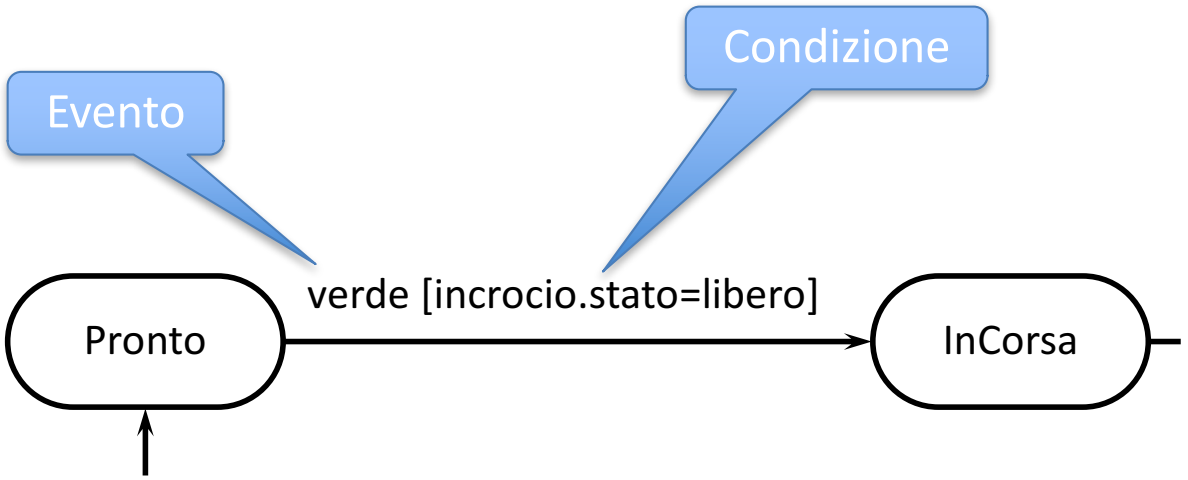
- Rappresentano il comportamento dei singoli oggetti di una classe in termini di
  - Eventi a cui gli oggetti (la classe) sono sensibili
  - Azioni prodotte
  - Transizioni di stato
    - Identificazione degli stati interni degli oggetti
- Possibilità di descrivere evoluzioni parallele
- Sintassi mutuata da StateChart (D. Harel)

Primo esempio

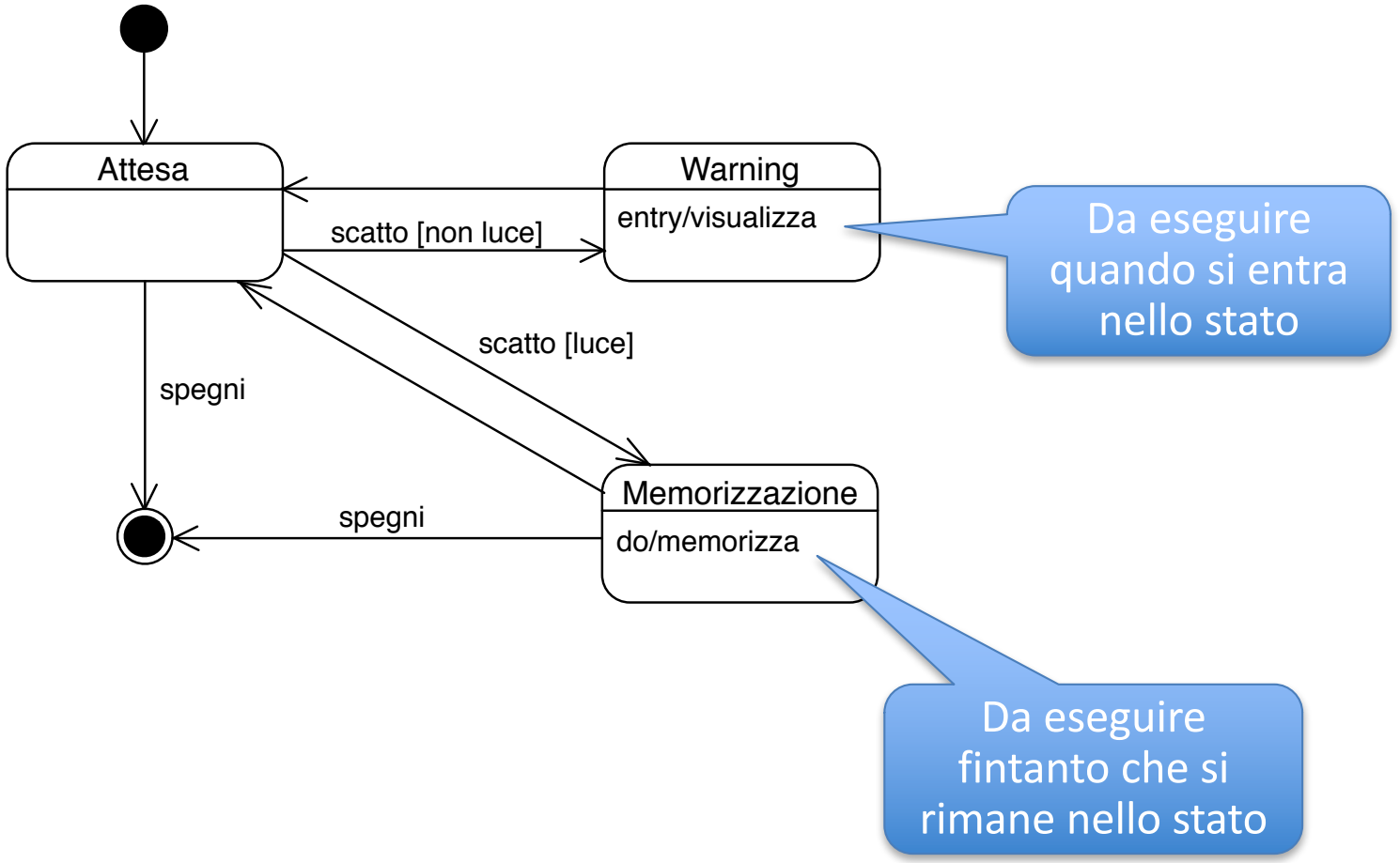


Eventi e condizioni

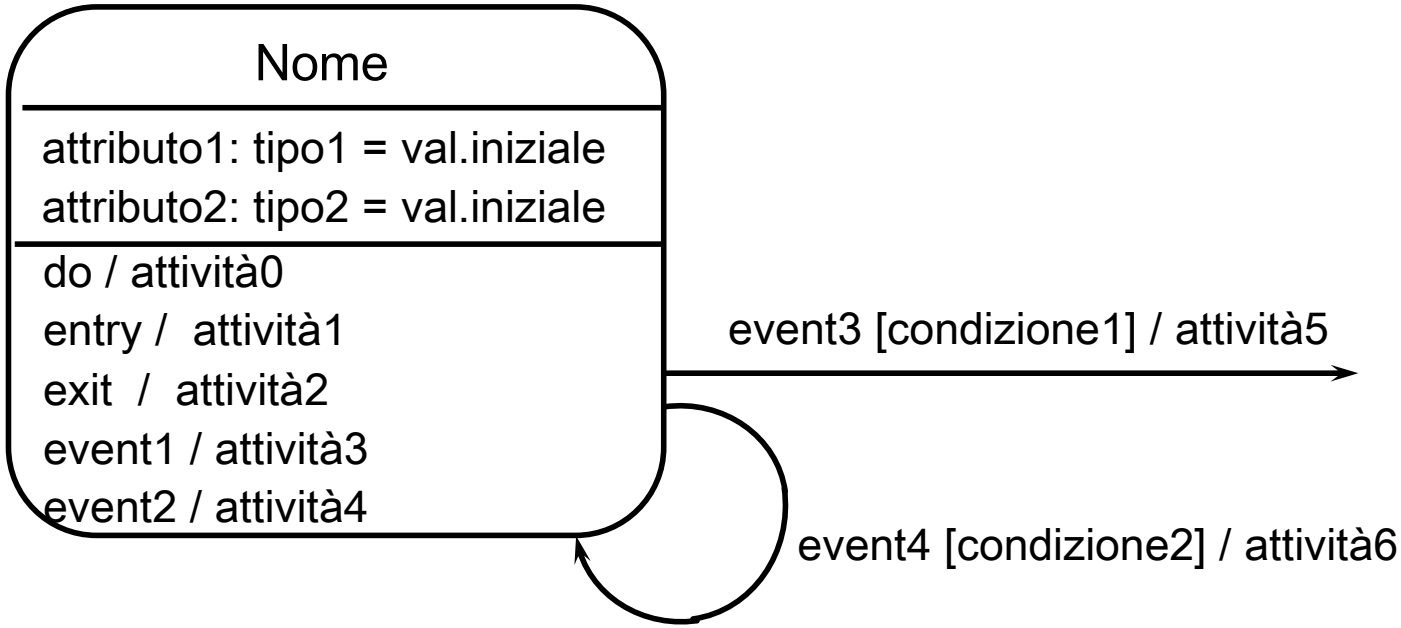
- Funzioni booleane sui valori degli oggetti
- Utili quando non basta l'evento, ma si vuole aggiungere un predicato



Aggiungiamo elementi

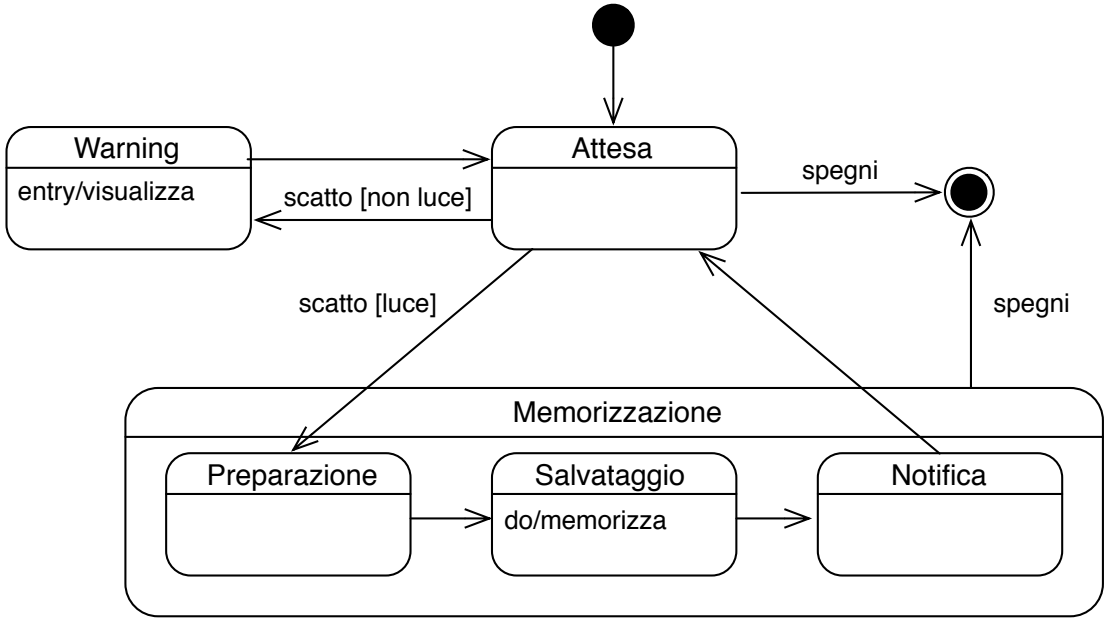


Stato completo



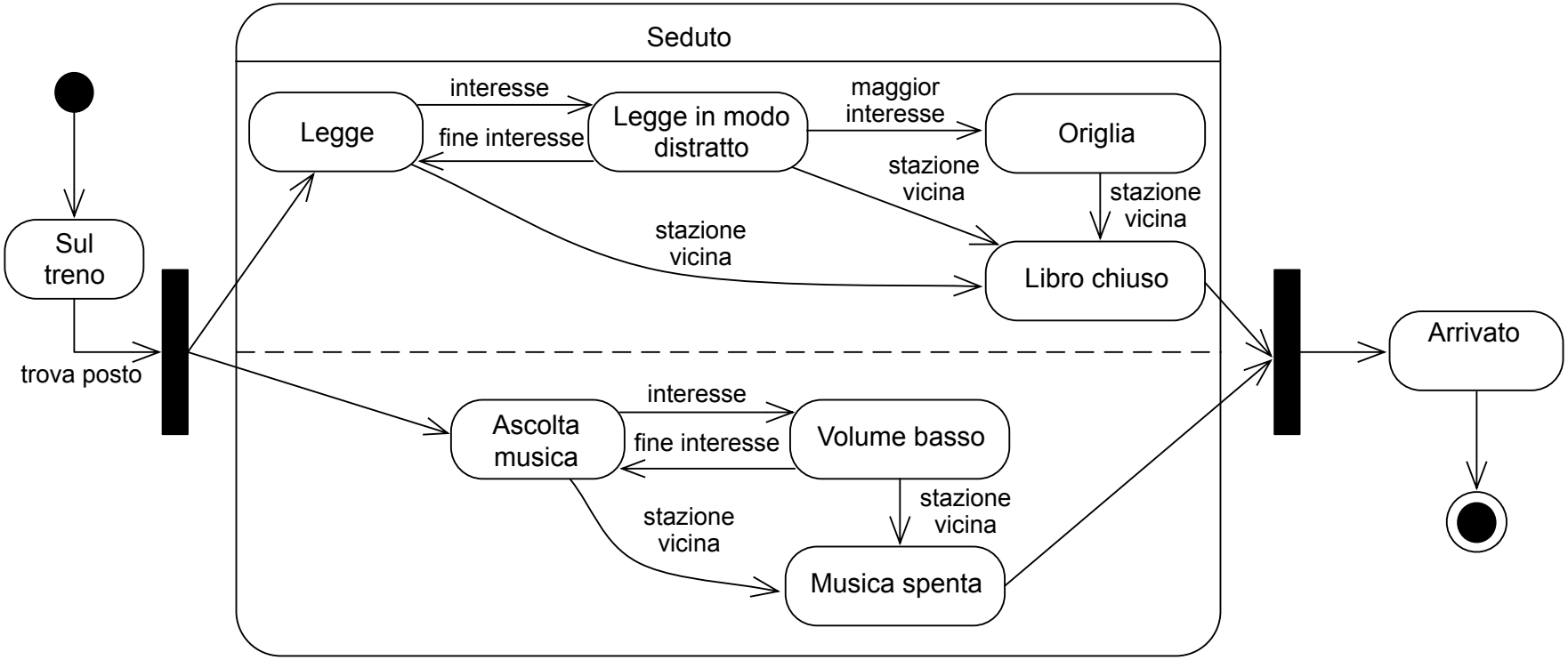
Decomposizione OR

- Un macro stato equivale ad una scomposizione OR degli stati
  - Solo uno degli stati costituenti può essere attivo in un certo istante
- I sottostati “ereditano” le transizioni dei loro superstati



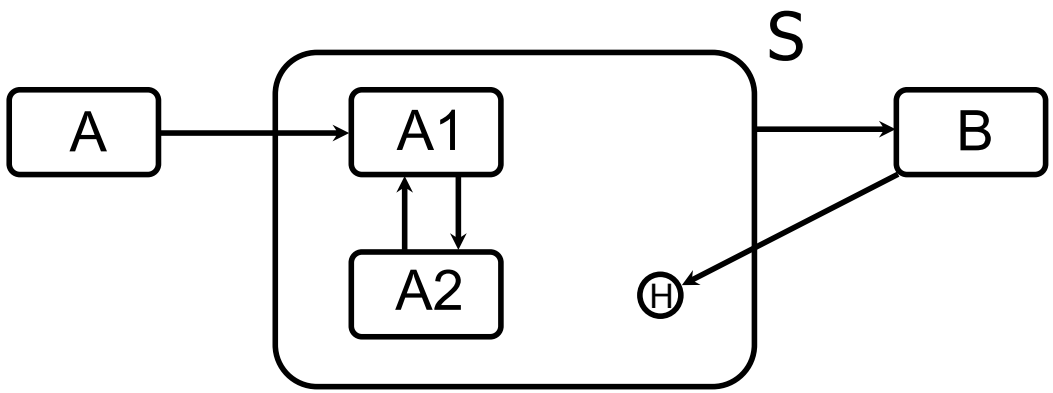
Decomposizione AND

- Duale rispetto al caso OR
  - Uno stato attivo per ciascun macrostato
  - Modella operazioni ed attività concorrenti



History

- History può essere associata a stati non foglia
- Quando l'esecuzione lascia uno stato S con history
  - Si salva l'ultimo stato visitato in S nella history H
- Quando l'esecuzione ritorna in S
  - Si riparte dall'ultimo stato salvato



Diagrammi dei componenti

- Utili per “decomporre” il sistema in esame

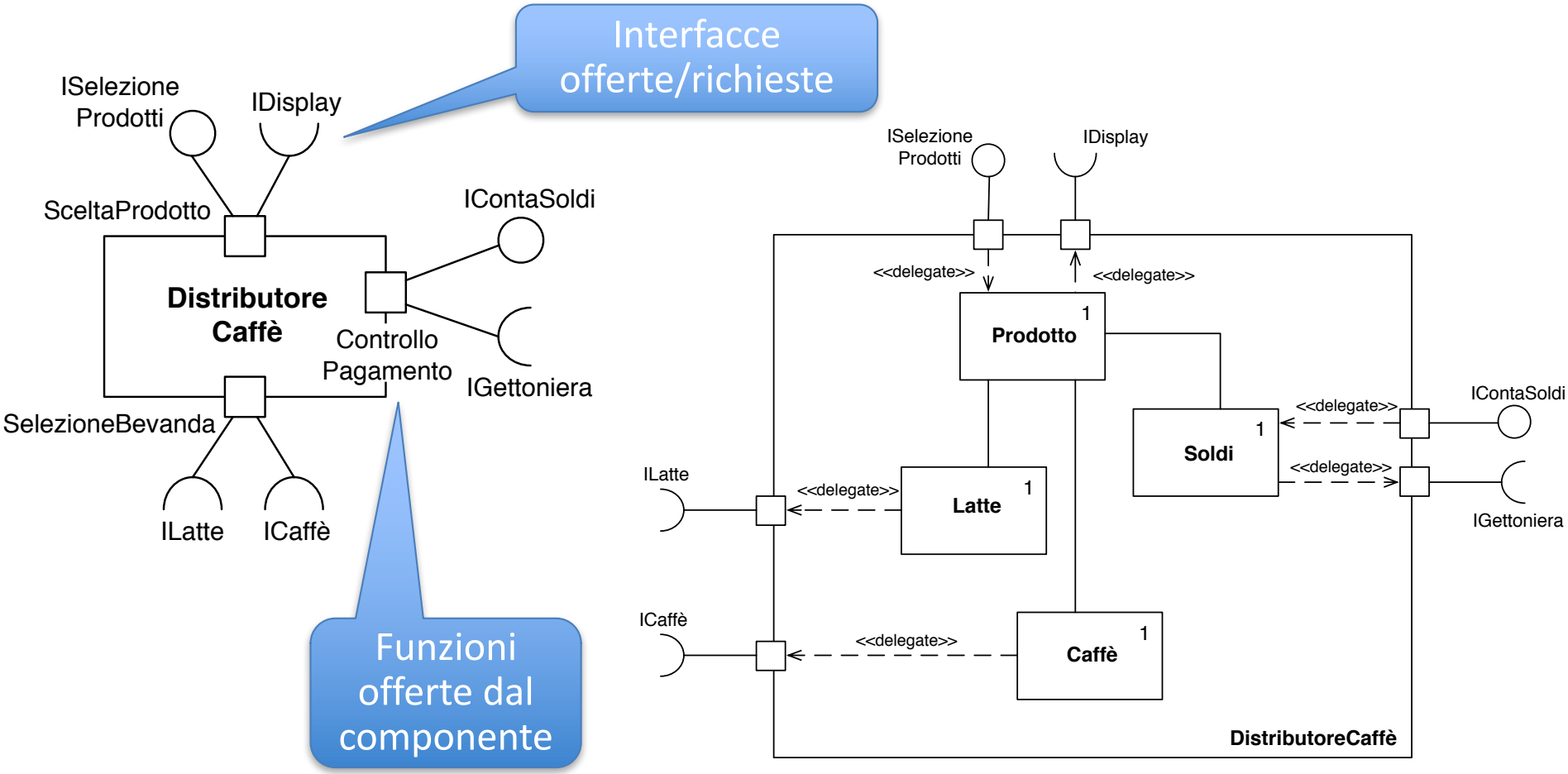


Diagramma di deployment