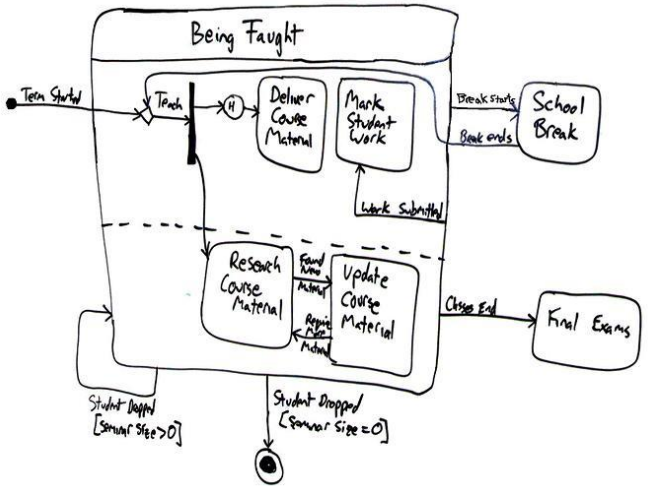


UML 2.0: STATE MACHINE DIAGRAM

Ingegneria del Software 2023-2024

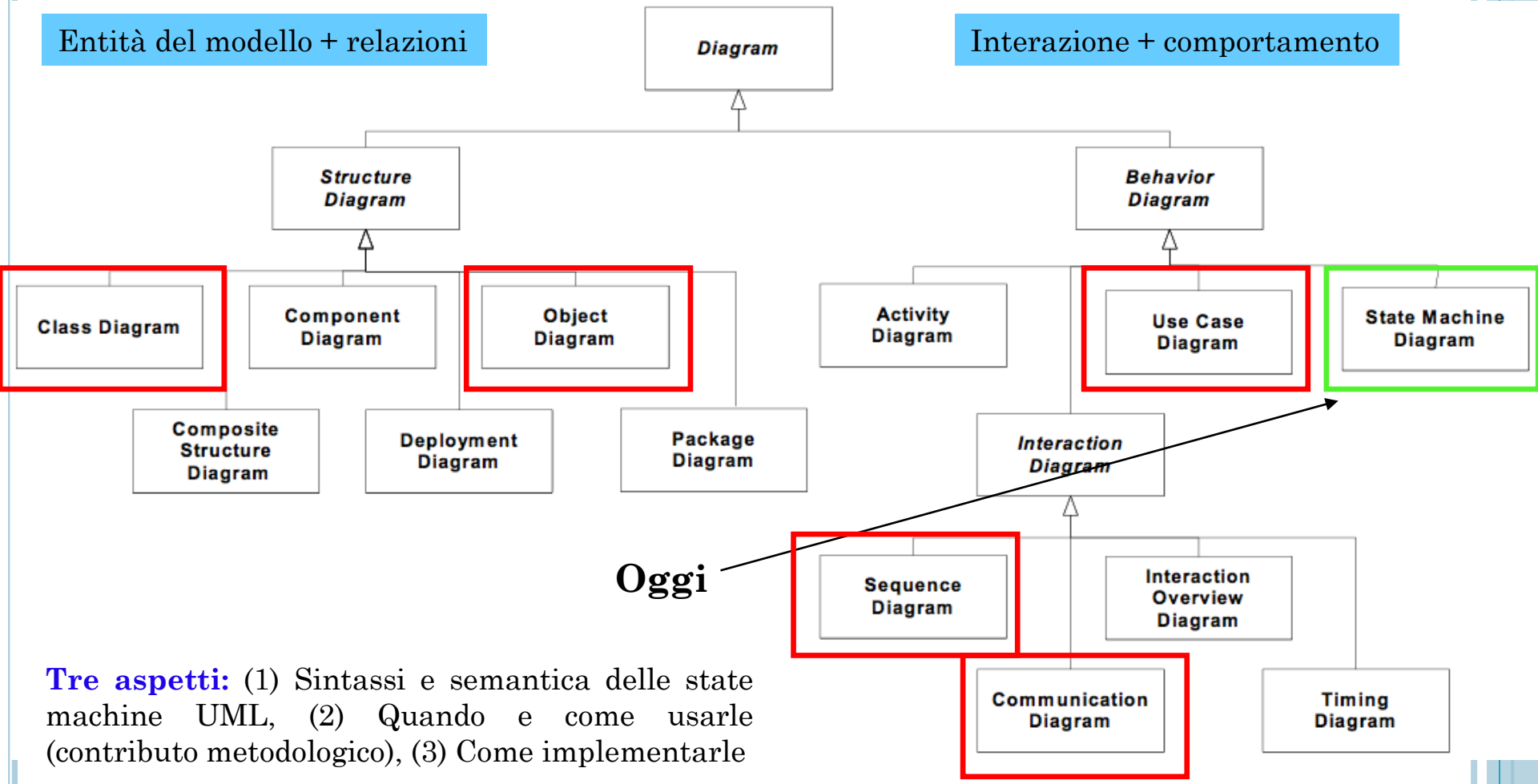
ORIENTIAMOCI

Modellazione statica

Entità del modello + relazioni

Modellazione dinamica

Interazione + comportamento



Tre aspetti: (1) Sintassi e semantica delle state machine UML, (2) Quando e come usarle (contributo metodologico), (3) Come implementarle

Diagrammi di macchine a stato

- **Cosa sono?**
  - Sono diagrammi che descrivono

  - Le state machine vengono usate per descrivere il **comportamento** di una **entità** come variazione del suo stato interno quando è sottoposta a sollecitazioni dal mondo esterno
    - **Entità** = sistema software, sistema hardware, classe OO, entità del mondo reale, …

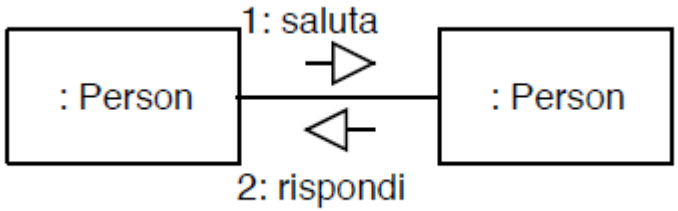
SEMBRA COMPLESSO VERO?

- Più semplicemente: le State machine descrivono cosa fa un entità (cioè come si comporta) quando riceve un input (evento)
- Non basta …
  - occorre ancora aggiungere che un entità può comportarsi in modo diverso a secondo del **suo stato interno**
- Es. **Distributore di caffè**

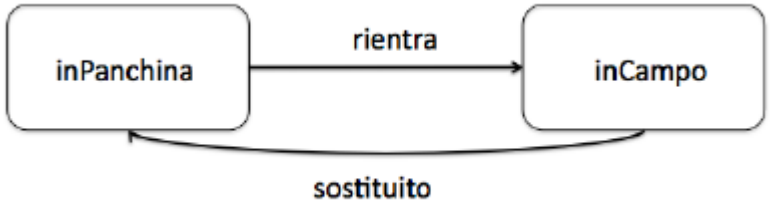
  - PushButton(eroga caffè) → eroga il caffè (stato=soldi inseriti OK)
  - PushButton(eroga caffè) → msg (stato=soldi inseriti insuff.)

Interazione vs. Macchina a stati

- **Diagramma di Interazione:** descrive un insieme di oggetti che si scambiano messaggi per raggiungere un dato obiettivo



- **Macchina a stati: descrive la sequenza di stati** in cui si trova un oggetto durante il suo ciclo di vita e in risposta a eventi

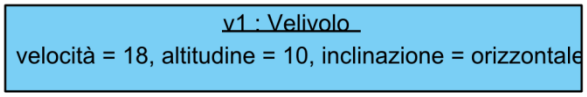


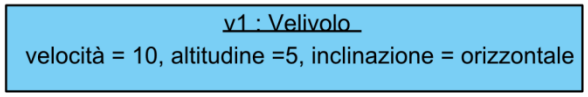
NOZIONE DI STATO

- Esempio: un velivolo può trovarsi nei seguenti 5 stati:
  - *On, Off, TakingOff, Landing, Flying*
- Rappresenta una situazione (nella vita di un oggetto) durante la quale delle **condizioni vengono soddisfatte** e delle **attività possono essere eseguite**
  - Stato Flying, inclinazione = orizzontale, attività → servire caffè

Warning:
Quando gli sviluppatori parlano di stato spesso intendono una combinazione di valori di tutti i campi di un oggetto.

Gli stati di una macchina a stati UML sono qualcosa di più astratto: ad ognuno di essi corrisponde un diverso comportamento del sistema al verificarsi degli eventi

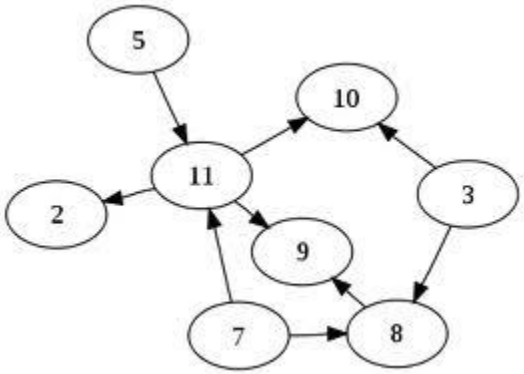




Stato è sempre Flying

Rappresentazione grafica

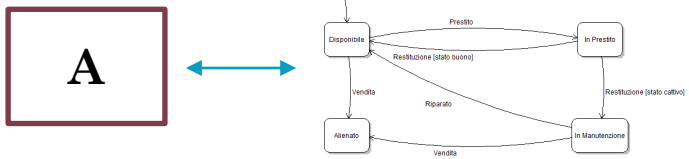
- Graficamente le state machine UML sono rappresentate come **grafi …**
  - nodi e archi

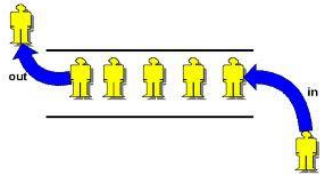


- La nozione di grafo è arricchita con sintassi specifica e semantica

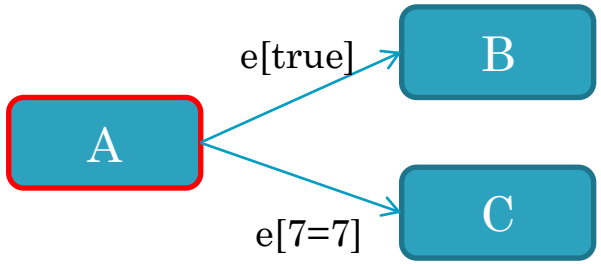
- In particolare :
  - **nodo**: stato
  - **arco**: transizione (evento + guardia + attività)

SEMANTICA



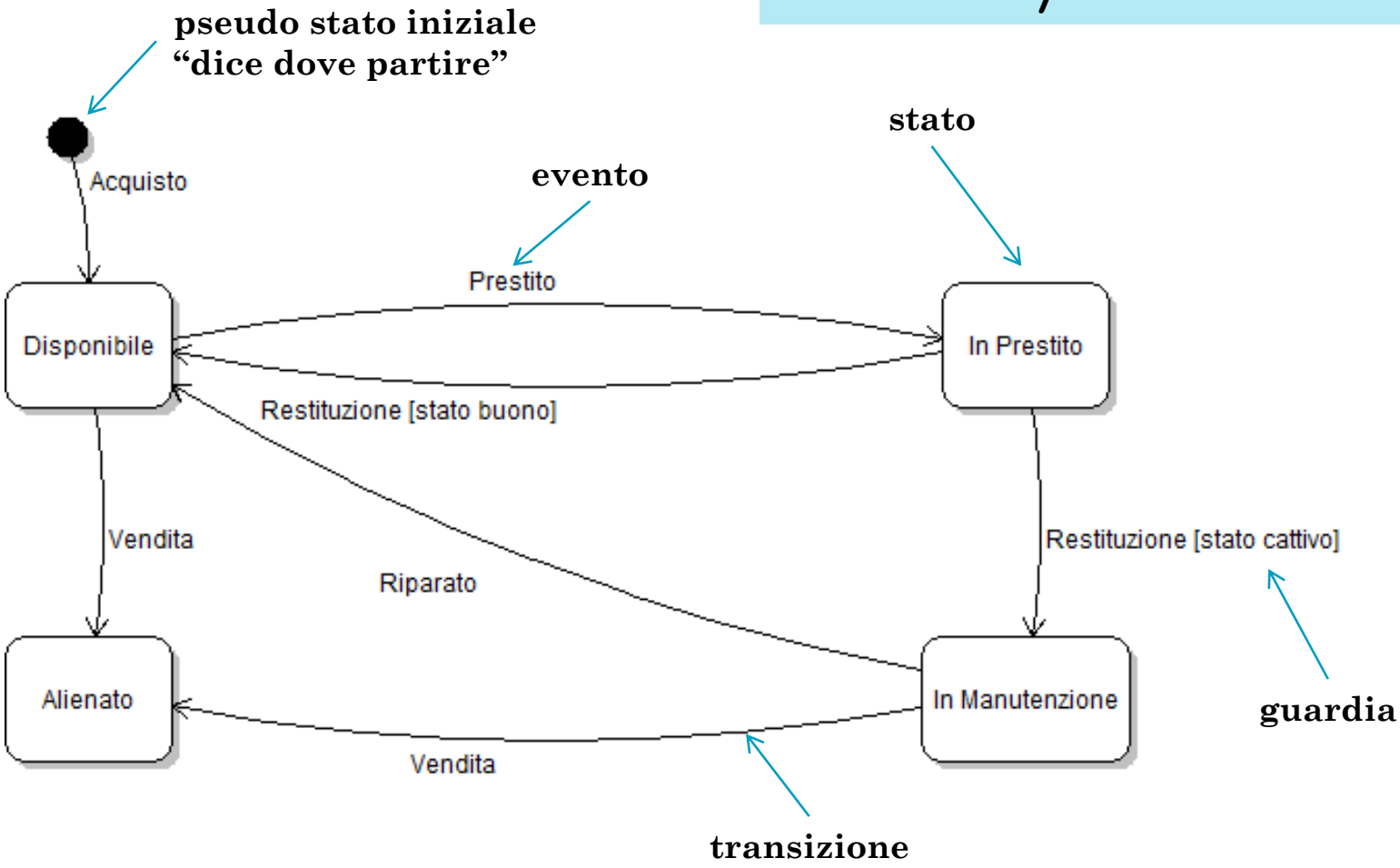


- Una macchina a stati è sempre associata ad un **entità** (es. classe o sistema SW) e ne descrive il suo **comportamento**
- La macchina a stati riceve eventi che vengono salvati su una coda ed estratti uno alla volta (~Buffer)
- La semantica di questi eventi è di tipo **run-to-completion:** un evento viene estratto solo dopo che il precedente è stato processato
- Se ci sono **più transizioni eseguibili** in un dato momento solo una viene eseguita
  - **in modo non deterministico**
    - es. due transizioni dallo stesso stato con lo stesso evento e due condizioni diverse, entrambe vere



Esempio Copia Libro (Biblioteca)

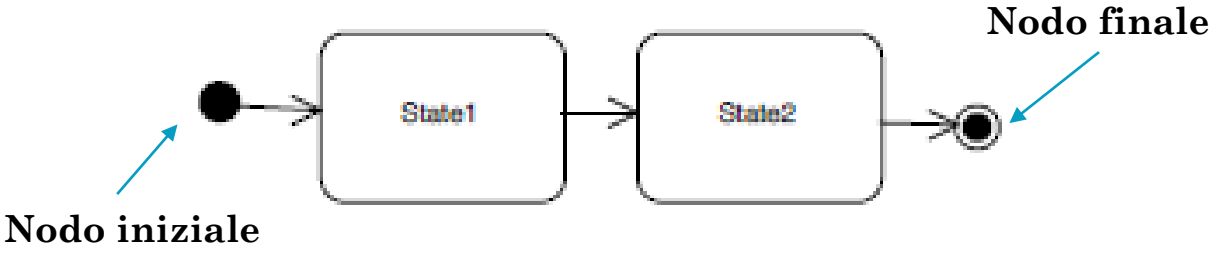
State machine rappresenta il ciclo di vita di una copia di un libro

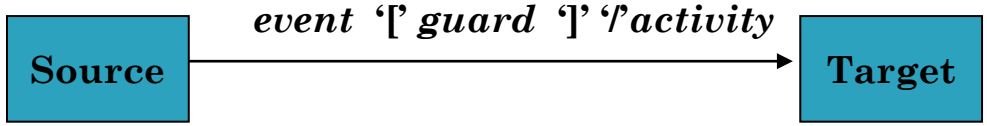


Prospettiva concettuale

Nodo iniziale e nodi finali

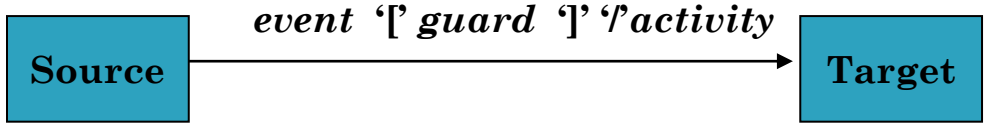
- Il disco nero marca l'**inizio** dell'esecuzione. Non è uno stato vero e proprio (pseudo-stato) ma un marcatore che punta allo stato da cui partire
  - **È unico** per ogni diagramma di stato
    - o stato composito
- Il disco nero bordato (**nodo finale**), indica che l'esecuzione è terminata
  - Possono comparire **in qualunque numero** all'interno di un diagramma





TRANSIZIONI (1)

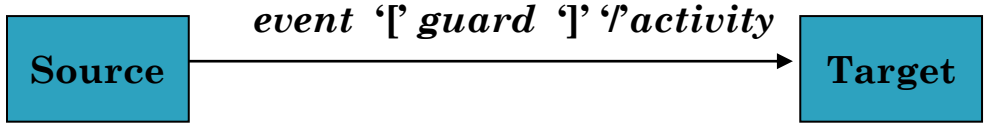
- Rappresentano come si passa da uno stato ad un altro
- Ogni transizione, oltre allo **stato origine** e **destinazione**, può specificare:
  - **Evento**: un ‘trigger’ che attiva il passaggio di stato
  - **Guardia**: **una condizione** che, se vera, permette il passaggio di stato
  - **Attività**: una o più **azioni** che sono compiute prima di cambiare stato
- Una transizione avviene come risposta ad un evento (se la guardia è vera), e al momento della transizione l’entità esegue l’**attività** specificata



TRANSIZIONI (2)

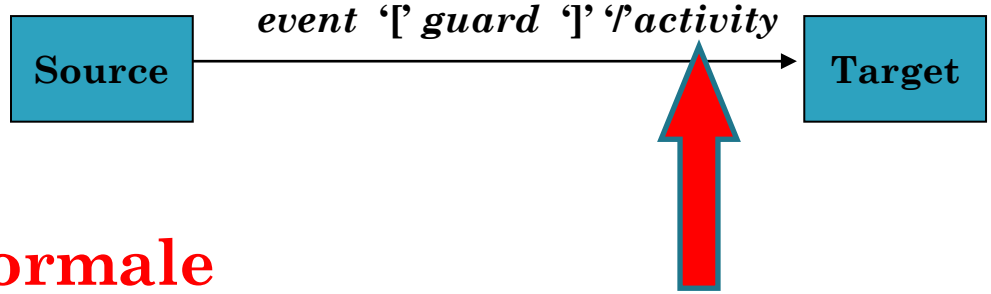
- Un **evento** è qualcosa che **l’entità subisce**, come ad esempio la ricezione di un messaggio
- Un attività (composta da + **azioni**) è qualcosa che **l’entità esegue**, come l’invio di un messaggio o l’esecuzione di un comando
  - Es. copiaLibro.setPresente(true)

    cont := cont +1;
- Una **guardia** può essere espressa:
  - in linguaggio naturale
  - in pseudo-linguaggio
  - con un linguaggio di programmazione (es. Java)
  - in OCL



SONO TUTTE OPZIONALI!

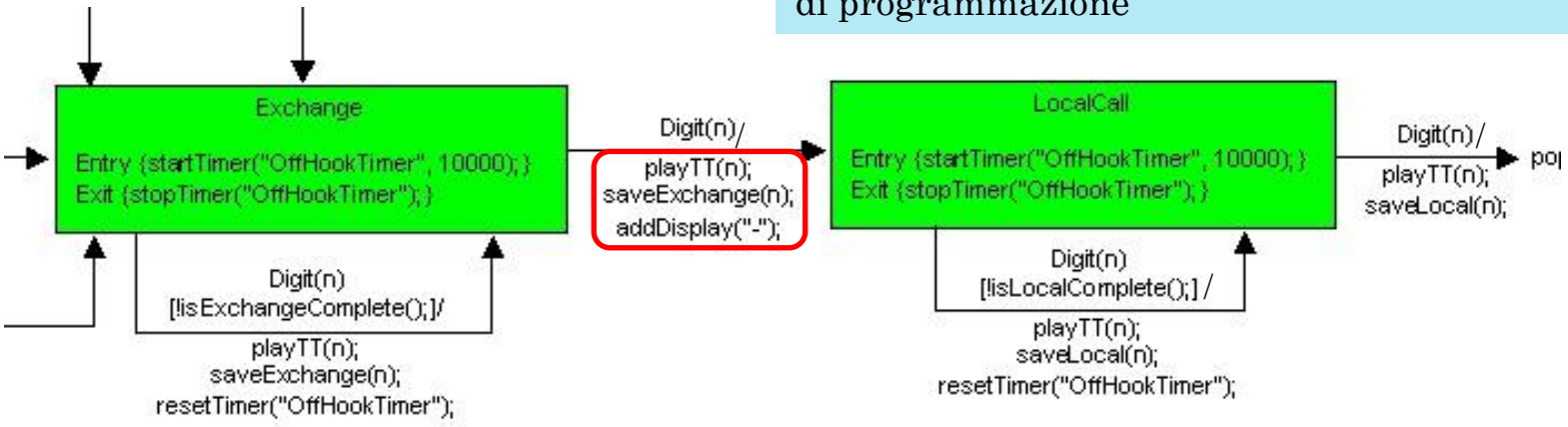
- Eventi, Guardie e Attività
  - Sono tutti **opzionali!**
- Se manca l’**attività** significa che durante la transizione si cambia stato ma non si fa altro
- La mancanza della **guardia** indica che la transizione è intrapresa ogni volta che si verifica l’evento
- Se manca l’**evento** vuol dire che la transizione avviene immediatamente



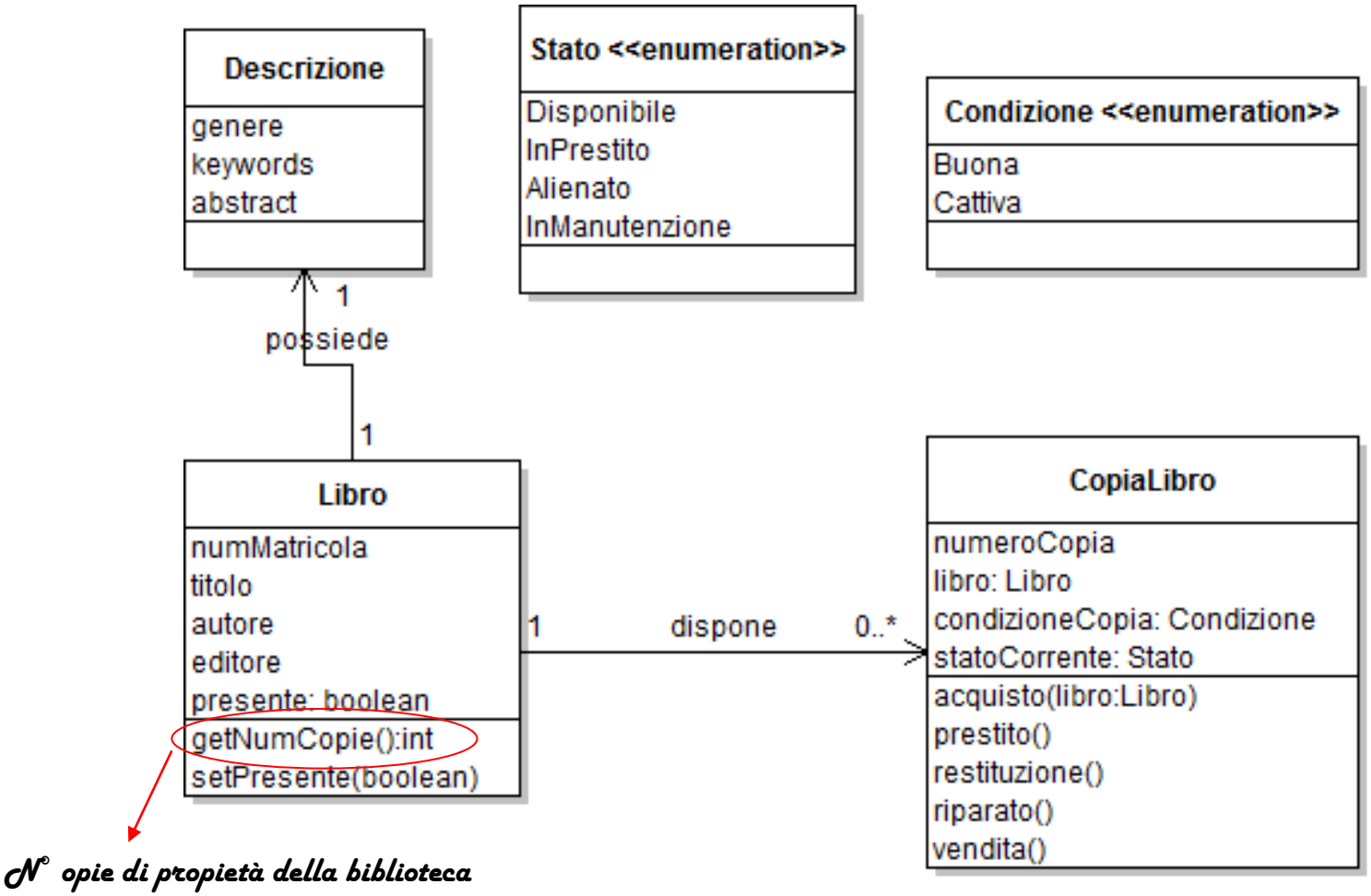
ATTIVITÀ

- Espresse in **modo informale**
  - Linguaggio naturale
    - Es. "aggiungi studente alla lista"
- Espresse sotto forma di **pseudolinguaggio**
  - Lista di comandi/azioni separati da ";"
    - Cambiare il valore di una variabile
    - Compiere un operazione I/O
    - Invocare un metodo
    - …

Esiste anche **Action Language** for Foundational UML (ALF) quando UML è usato come linguaggio di programmazione



Esempio 'copia libro' (Class diagram)

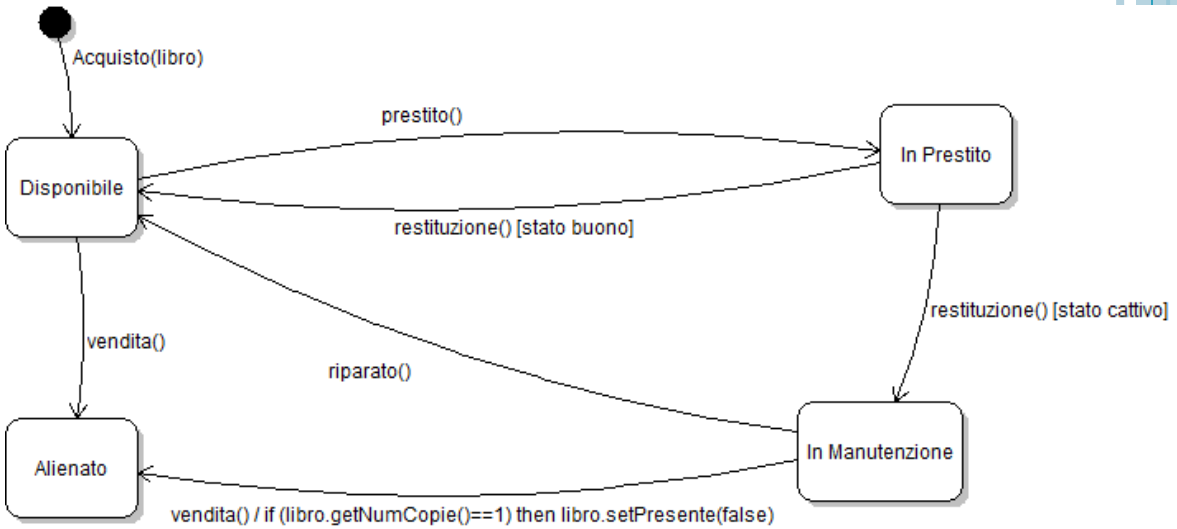


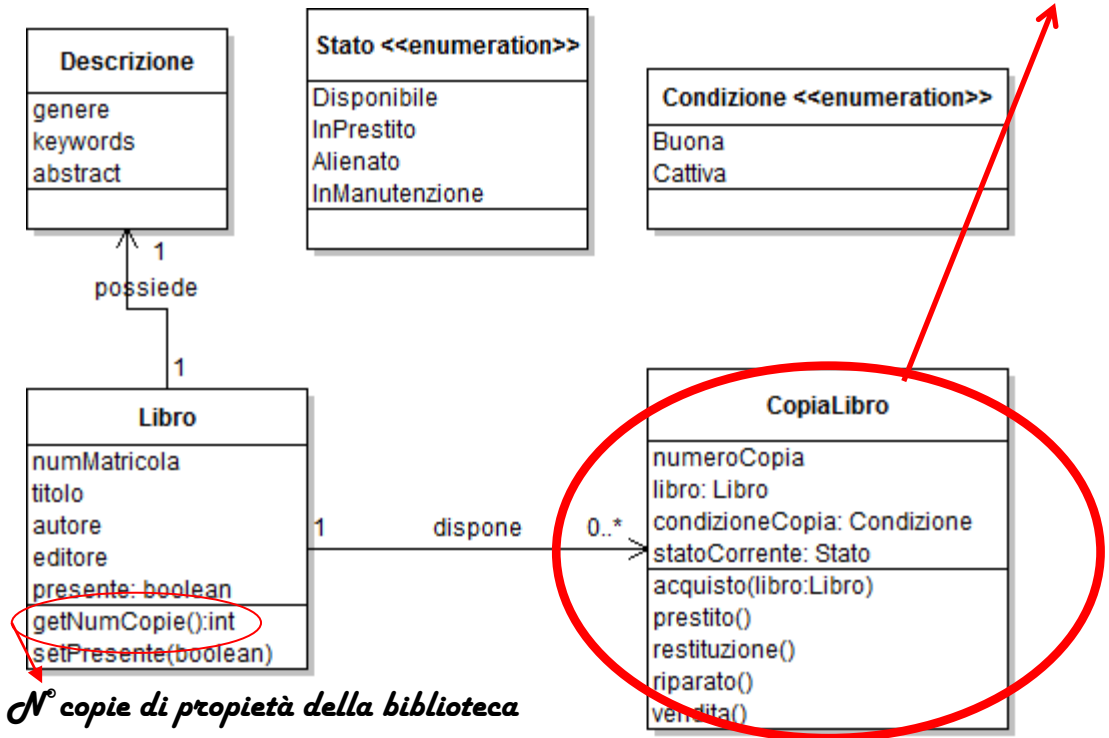
Prospettiva software

ESEMPIO 'COPIA LIBRO' (STATE MACHINE)

Una state machine è:

- **associata a una classe**
- mostra il comportamento di un singolo oggetto per la durata del suo ciclo di vita





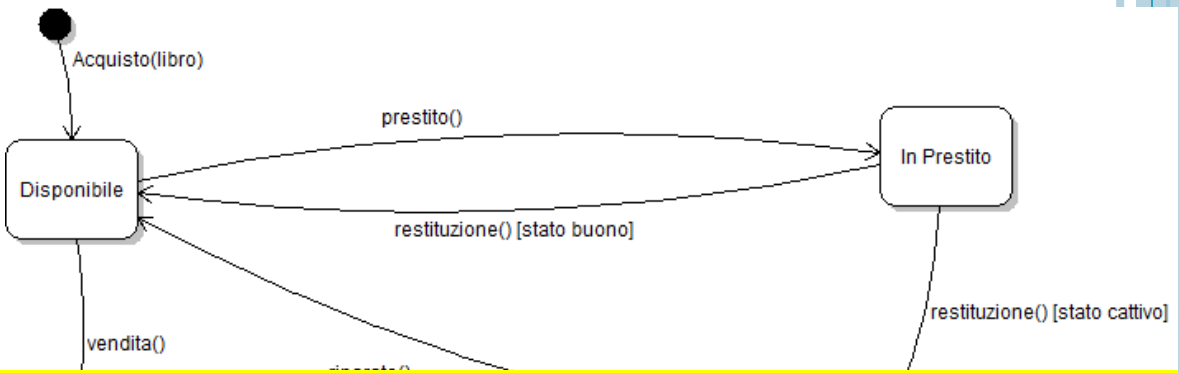
Attività che specifica che se la copia era l'unica allora viene settato l'attributo presente di Libro a false; non abbiamo più copie del libro in biblioteca

Prospettiva software

Esempio 'copia libro' (State machine)

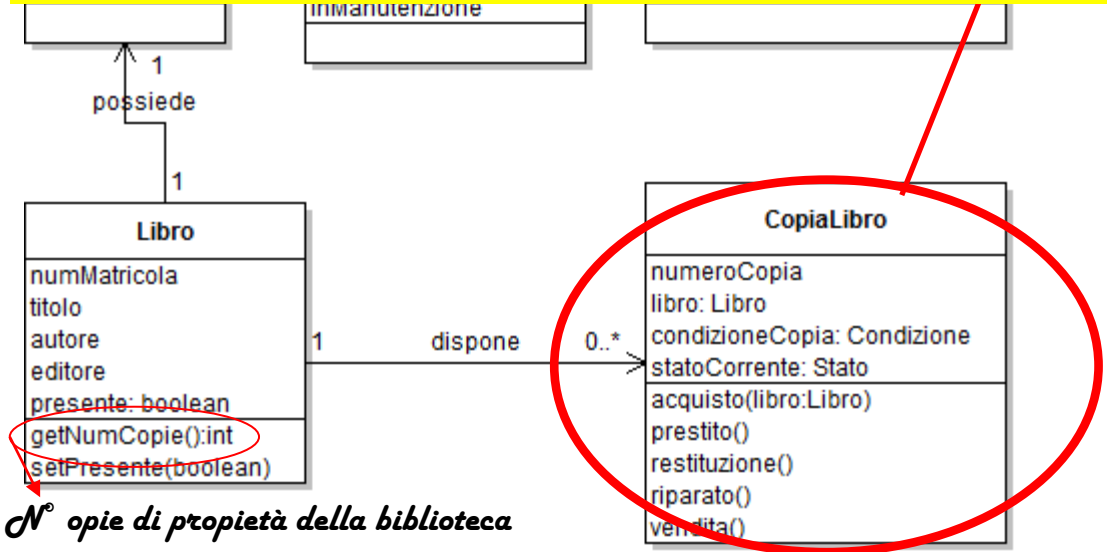
Una state machine è:

- **associata a una classe**
- mostra il comportamento di un singolo oggetto per la durata del suo ciclo di vita



Attenzione:
le azioni **non possono far riferimento a "cose" che l'oggetto non conosce**

In un azione possiamo sfruttare attributi, operazioni e link dell'oggetto che stiamo descrivendo, nonchè tutti i parametri del messaggio che ha fatto scattare la transizione

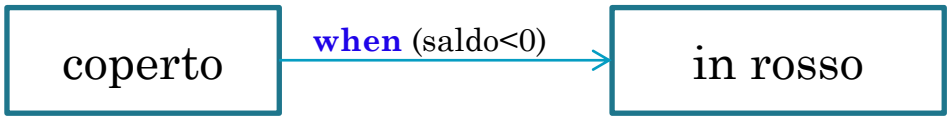


Se la copia era l'unica allora setta l'attributo presente di libro a false; non abbiamo più copie del libro in biblioteca

Prospettiva software

ALTRI TIPI DI EVENTI IN UML ...

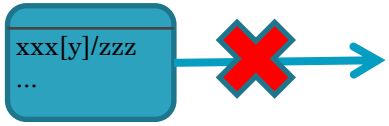
- Fino adesso per noi un evento è **solo la ricezione di un messaggio ...**
- Ma nelle State machine esistono diversi tipi di eventi, ad esempio:
  - **Evento di cambiamento**: si verifica quando una condizione passa da falsa a vera



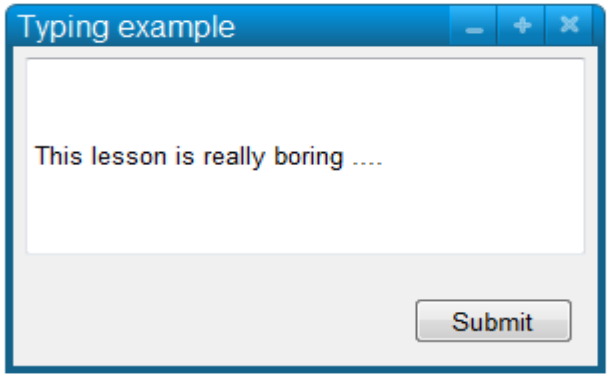
Conto corrente

  - **Evento temporale**: si verifica dopo un certo tempo o ad una data/ora "precisa"
    - **After** (2 minuti)
    - **At** (24:00)

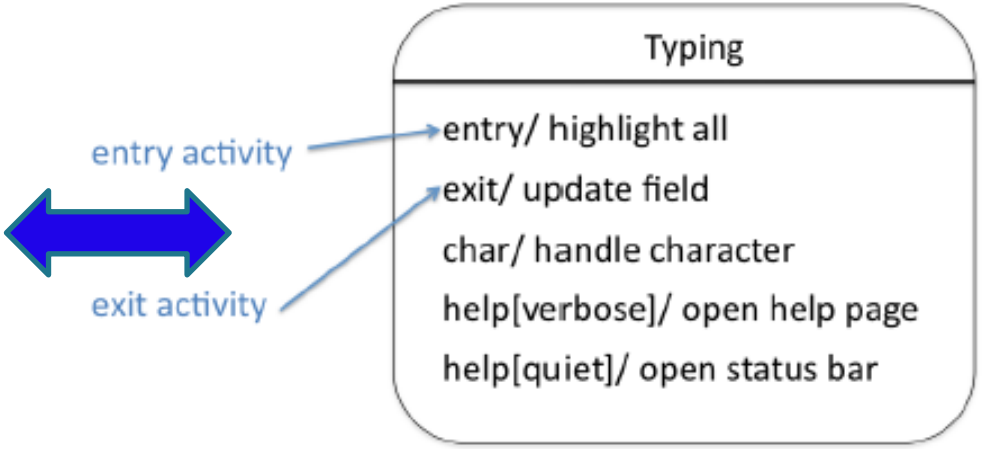
ATTIVITÀ INTERNE



- Uno stato può **reagire ad eventi** e compiere **attività** anche senza una transizione ad uno stato diverso
- Le **attività interne** sono mostrate nel secondo slot e seguono la stessa sintassi delle transizioni
  - **'entry' e 'exit'**: sono **attività interne speciali** che si attivano ogni volta che si entra o si esce dallo stato



Finestra testuale



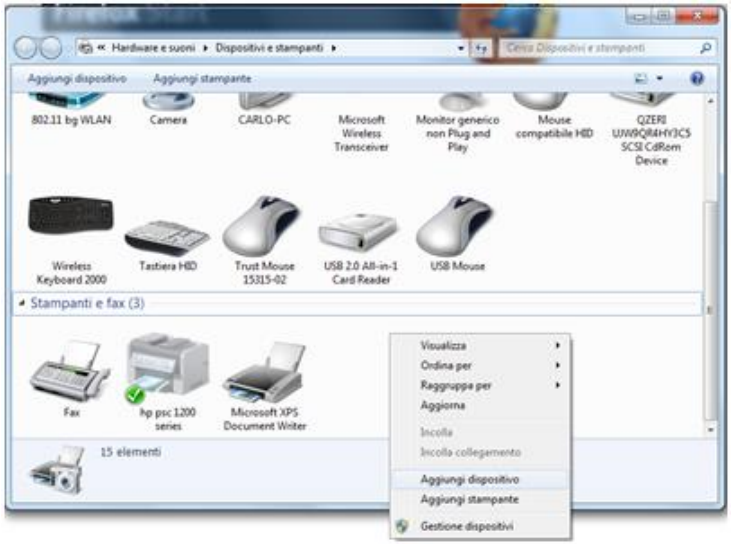
Attività interna simile (ma non uguale) ad una self-transition ....

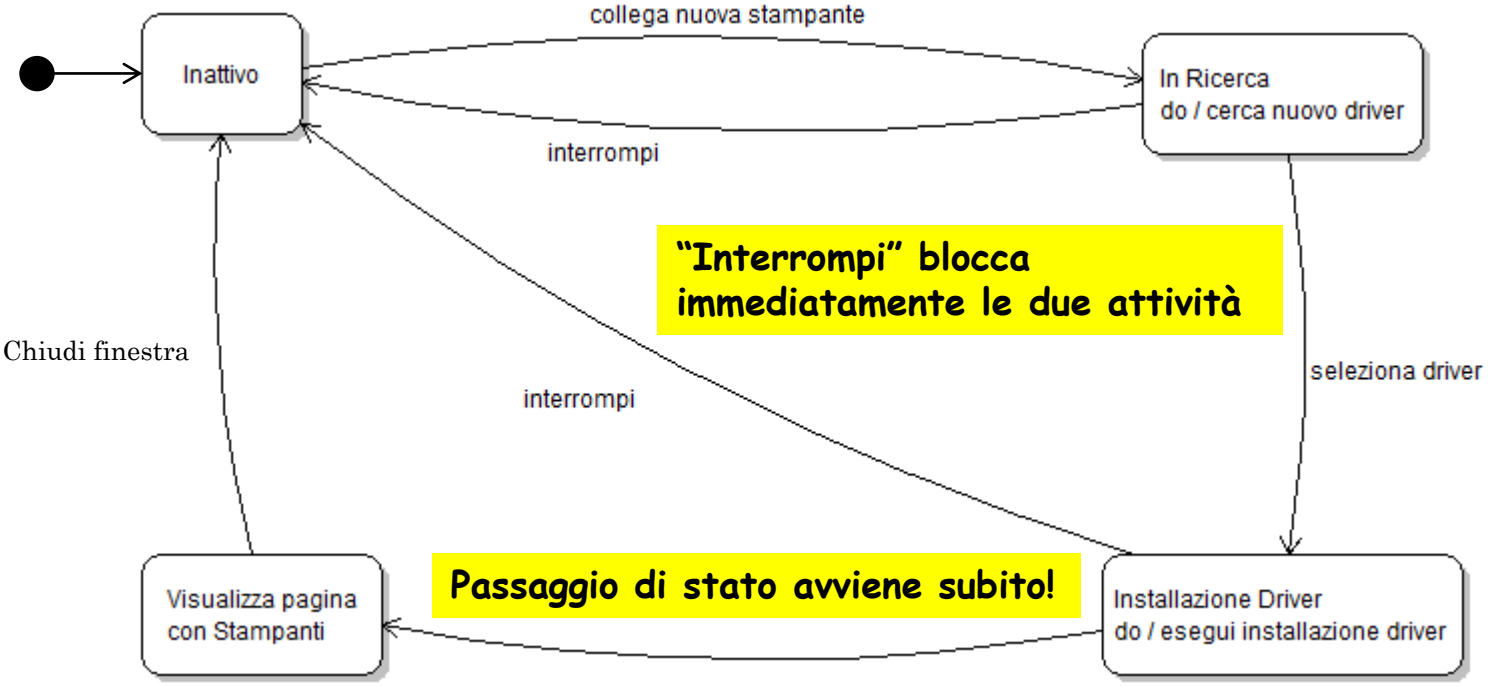
ENTRY, EXIT AND DO-ACTIVITY

- Una **self-transition** attiva sempre le attività 'entry' ed 'exit', le attività interne no
- All'interno di uno stato si possono usare alcune **attività speciali**, indicate tramite:
  - **entry:** eseguita quando l'oggetto **entra** nello stato
  - **exit:** eseguita quando l'oggetto **esce** dallo stato
  - **do (do-activity):** eseguita mentre l'oggetto **è nello stato**
- Una **do-activity** non è "istantanea" ma può durare per un intervallo di tempo ed essere interrotta da altri eventi
  - Le altre attività (sia normali che speciali) invece sono sempre atomiche e sono "istantanee"

ESEMPIO DO-ACTIVITY

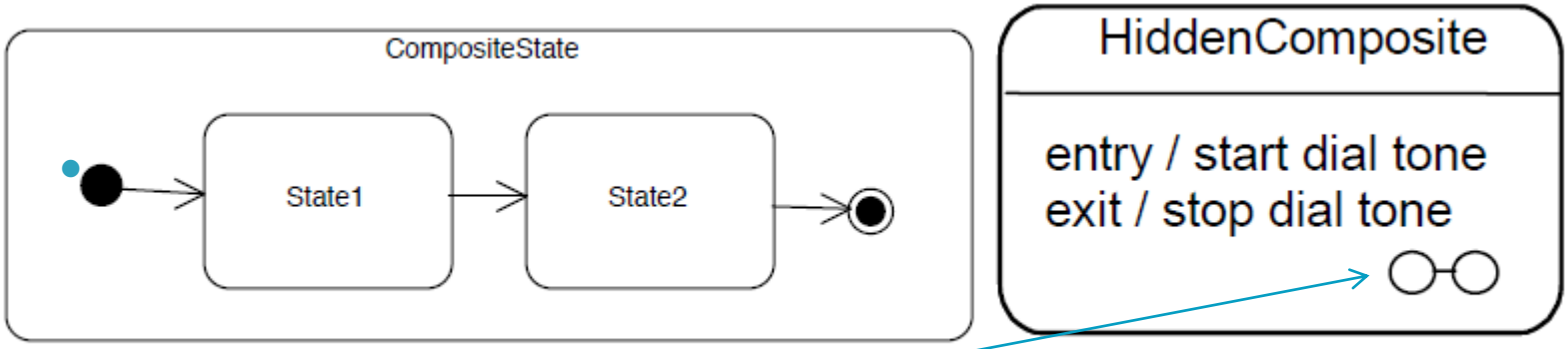
- Rappresentare con una State machine UML il processo di installazione di una stampante su un PC





STATI COMPOSITI (SUPERSTATI)

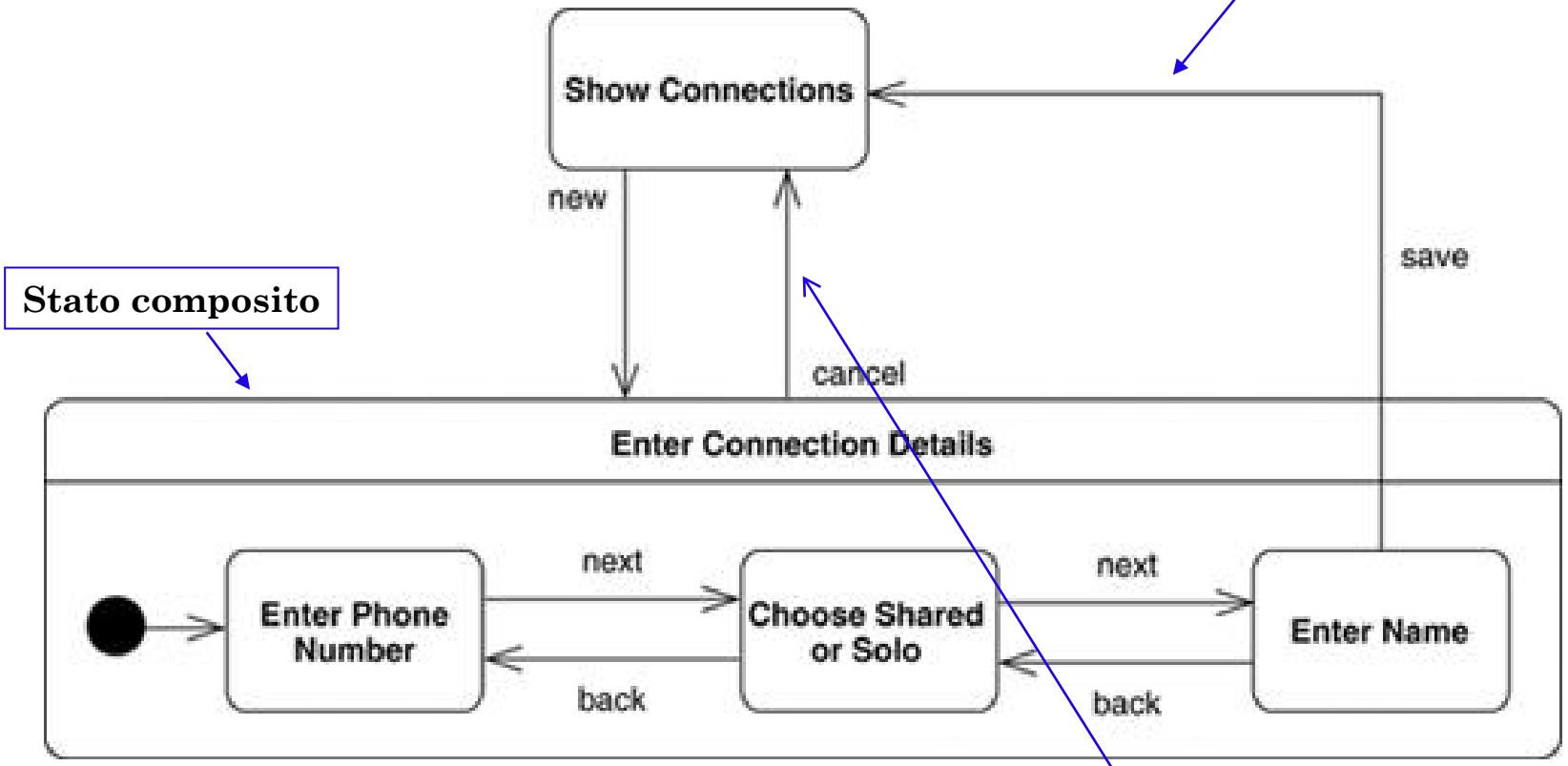
- Uno stato può essere **semplice** oppure **composito** (composto)
- Uno stato composito ci permette di **suddividere la complessità del modello:**
  - dall'esterno si vede un macro-stato, al cui interno vi sono altri stati (sotto-stati)



- Si può usare un **icona** per rappresentare uno stato composito il cui **comportamento interno non è mostrato**

STATI COMPOSITI: ESEMPIO

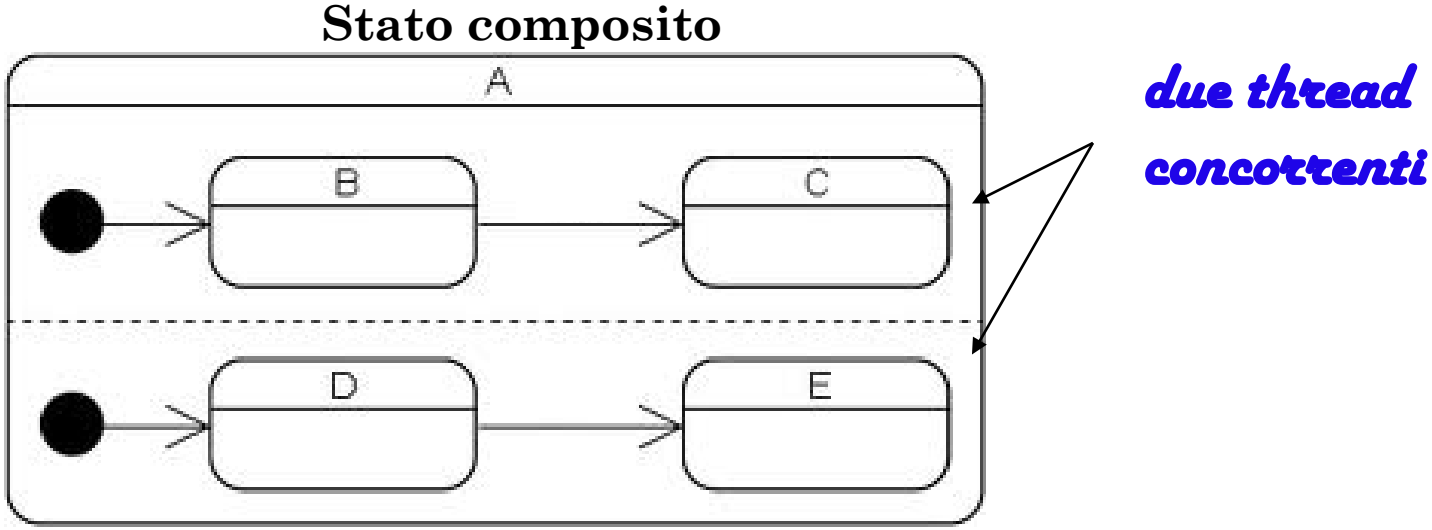
Si possono definire transizioni da uno stato interno di uno stato composito verso stati esterni



Transizioni in uscita dal bordo dello stato composito sono ereditate da tutti gli stati all'interno

STATI COMPOSITI E CONCORRENZA

- Gli stati compositi sono utili **anche** per modellare **la concorrenza**
- Si divide lo stato composito in (sotto-)diagrammi **ortogonali** che **sono eseguiti concorrentemente**



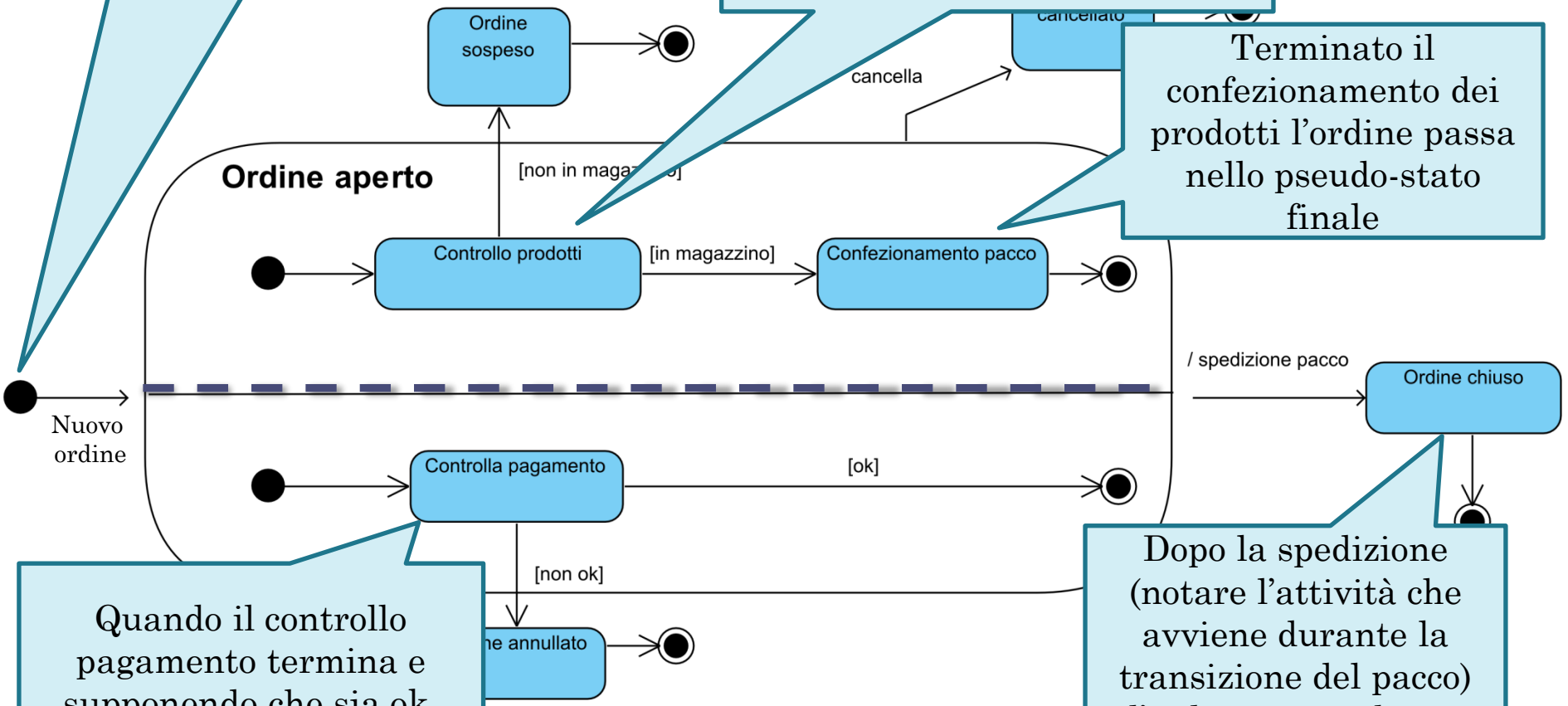
- L'entità rappresentata è al tempo stesso in uno stato della regione superiore (B o C) ed uno nella regione inferiore (D o E)

C MPIO

Appena arriva un nuovo ordine si apre l'ordine e l'ordine "passa" in Controllo prodotti (se presenti in magazzino) + Controllo pagamento

Se il controllo prodotti finisce per primo e tutti i prodotti sono in magazzino, allora l'ordine passa nello stato Confezionamento pacco

Terminato il confezionamento dei prodotti l'ordine passa nello pseudo-stato finale



Quando il controllo pagamento termina e supponendo che sia ok, l'ordine passa nello pseudo-stato finale

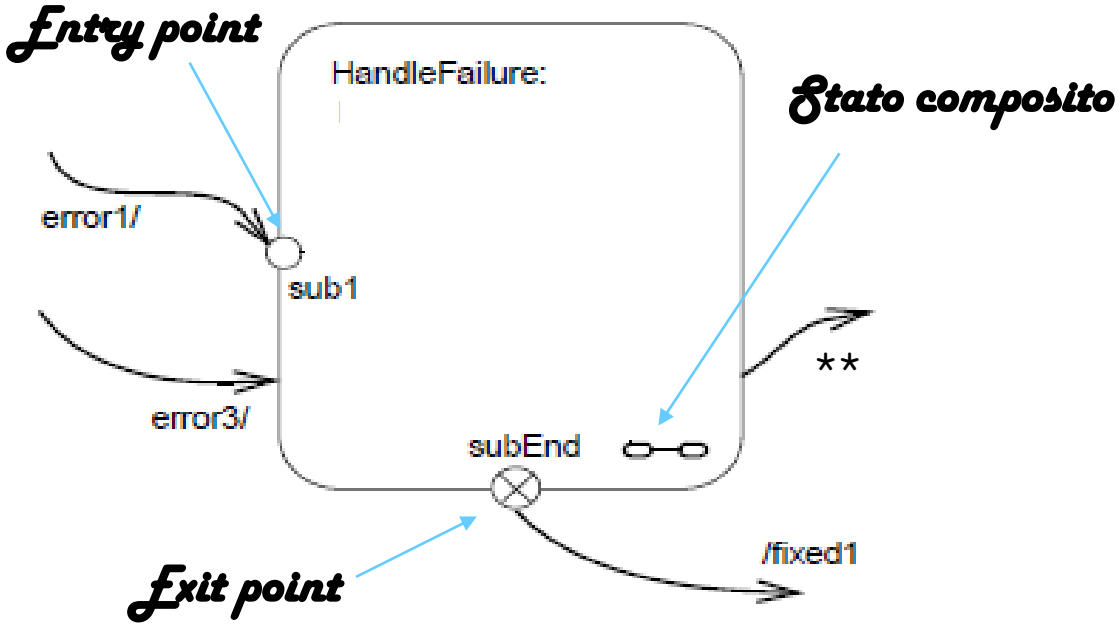
Dopo la spedizione (notare l'attività che avviene durante la transizione del pacco) l'ordine viene chiuso

Ciclo di vita di un ordine ...

Quando l'ordine lascia gli stati concorrenti è in un solo stato alla volta!

"Modellano entrata e uscita in punti diversi dal nodo iniziale e finale"

ENTRY E EXIT POINTS (STATI COMPOSITI)



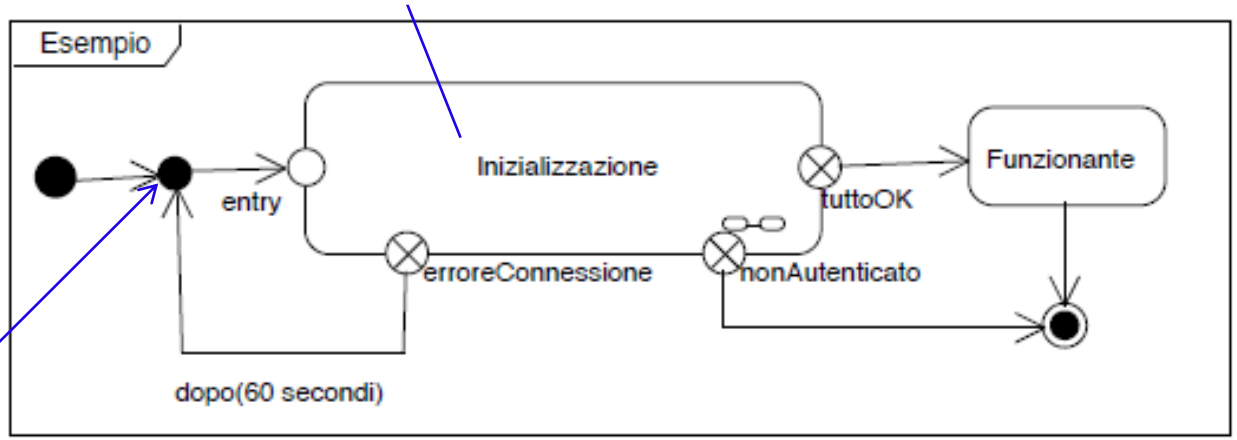
- L'evento "**error3**" fa partire l'esecuzione dallo stato iniziale dello stato composito
- L'evento "**error1**" fa partire l'esecuzione dall'entry point **sub1**
- Se l'esecuzione termina nell'exit point **subEnd** si esegue la transizione che genera l'attività **fixed1**
- Se l'esecuzione termina nello stato finale si segue la transizione sulla destra (**)

Entry e Exit Points: Esempio

Entry point

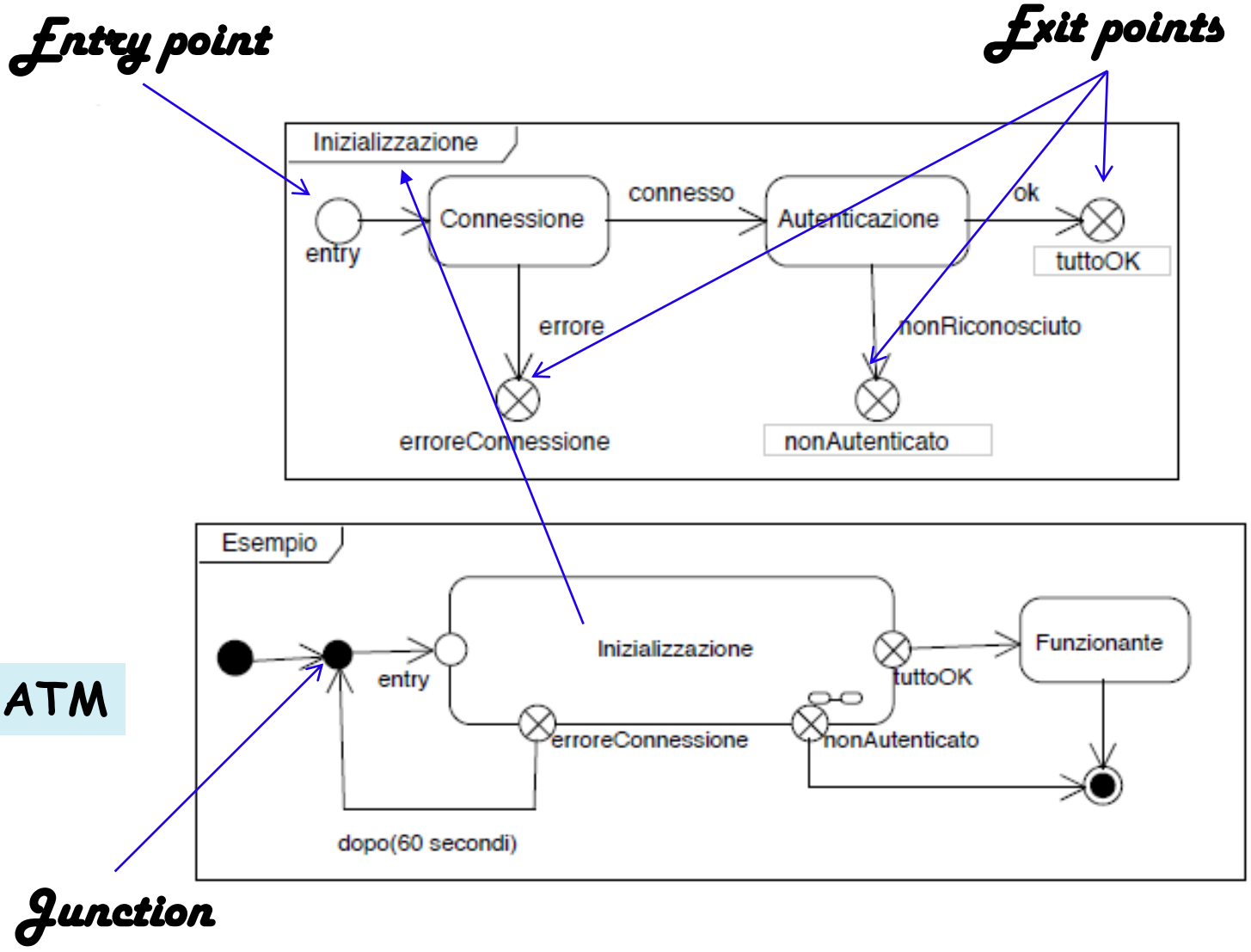
Exit points

ATM



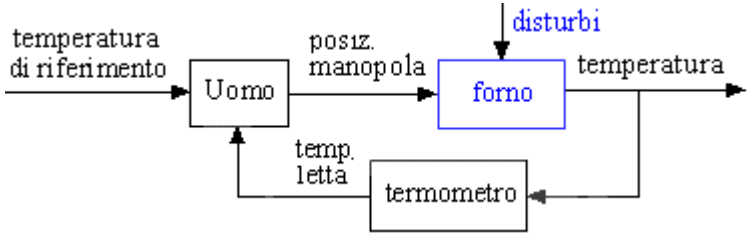
Junction

Entry e Exit Points: Esempio



Quando e dove usare le state machine?

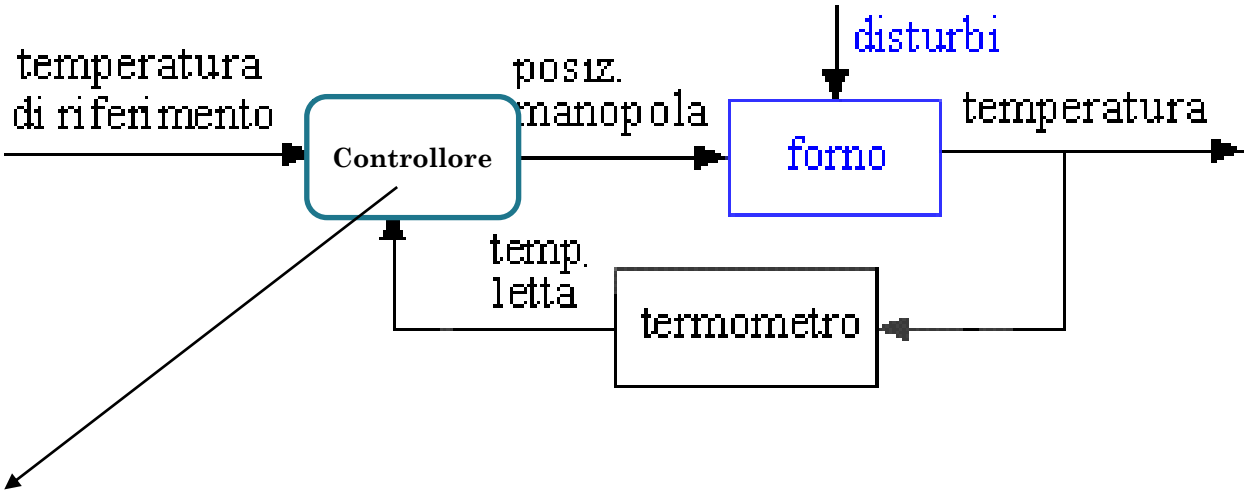
- Usare i diagrammi degli stati solo per le **entità che hanno una logica interna interessante e complessa!**



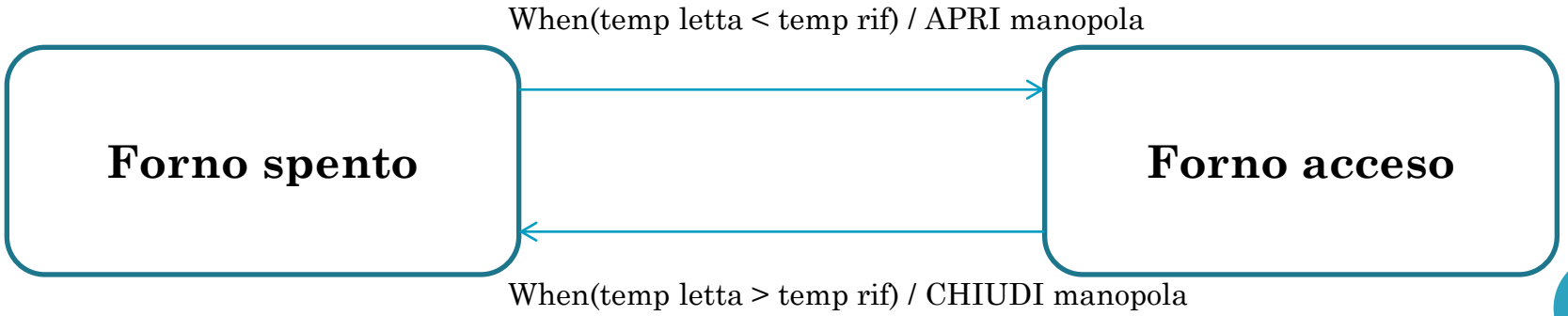
- Ad esempio:
  - Oggetti e sistemi di controllo
    - Es. Controllore automatico di un forno, controllore di volo, controllori di apparecchi medicali
      - Sistemi software/hardware capaci di 'pilotare' un sistema
  - Distributori automatici
    - Es. Distributore di bevande
  - Sistemi di gestione documentale
    - Molto usati nelle pubbliche amministrazioni
  - GUI
  - ...

FORNO (ESEMPIO)

Obiettivo del Controllore: tenere la temperatura del forno uguale a quella di riferimento



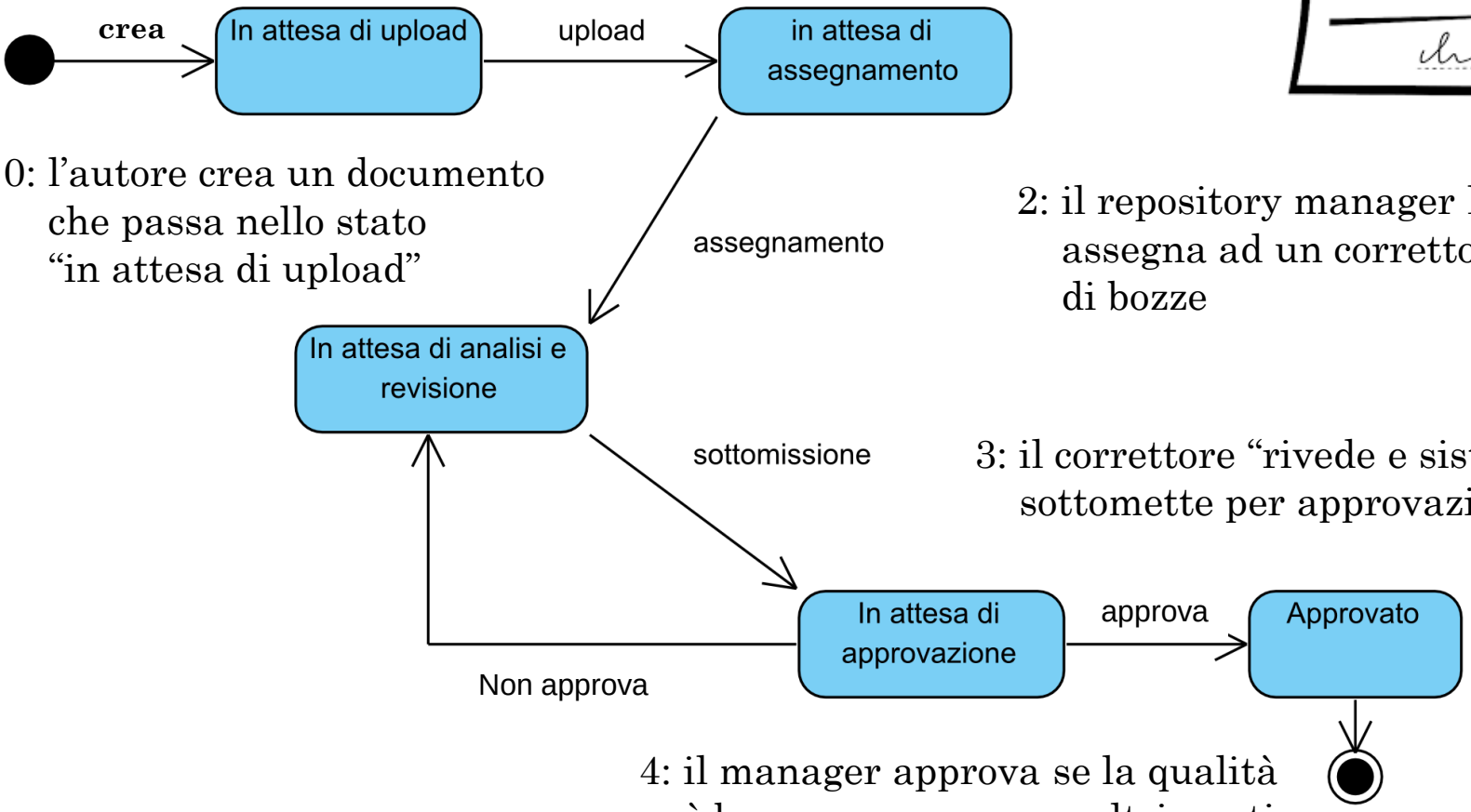
Controllore (sistema software che "pilota" la manopola)



State machine

STATI DI UN DOCUMENTO (ESEMPIO)

1: la segretaria fa l'upload sul sito Web



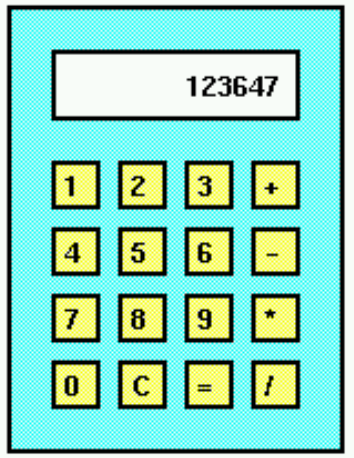
0: l'autore crea un documento che passa nello stato "in attesa di upload"

2: il repository manager lo assegna ad un correttore di bozze

3: il correttore "rivede e sistema" e sottomette per approvazione

4: il manager approva se la qualità è buona, non approva altrimenti

CALCOLATRICE (GUI ESEMPIO)



- All'inizio sono digitate delle cifre
- Successivamente viene digitata l'operazione
- Altre cifre
- Uguale (=)
  - Oppure altra "op" se si vuole calcolare un'espressione
- e viene stampato il risultato ….

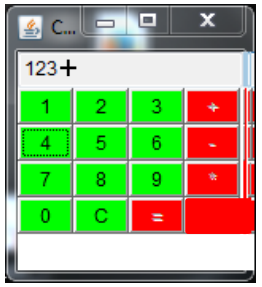
123635 + 12 = 123647

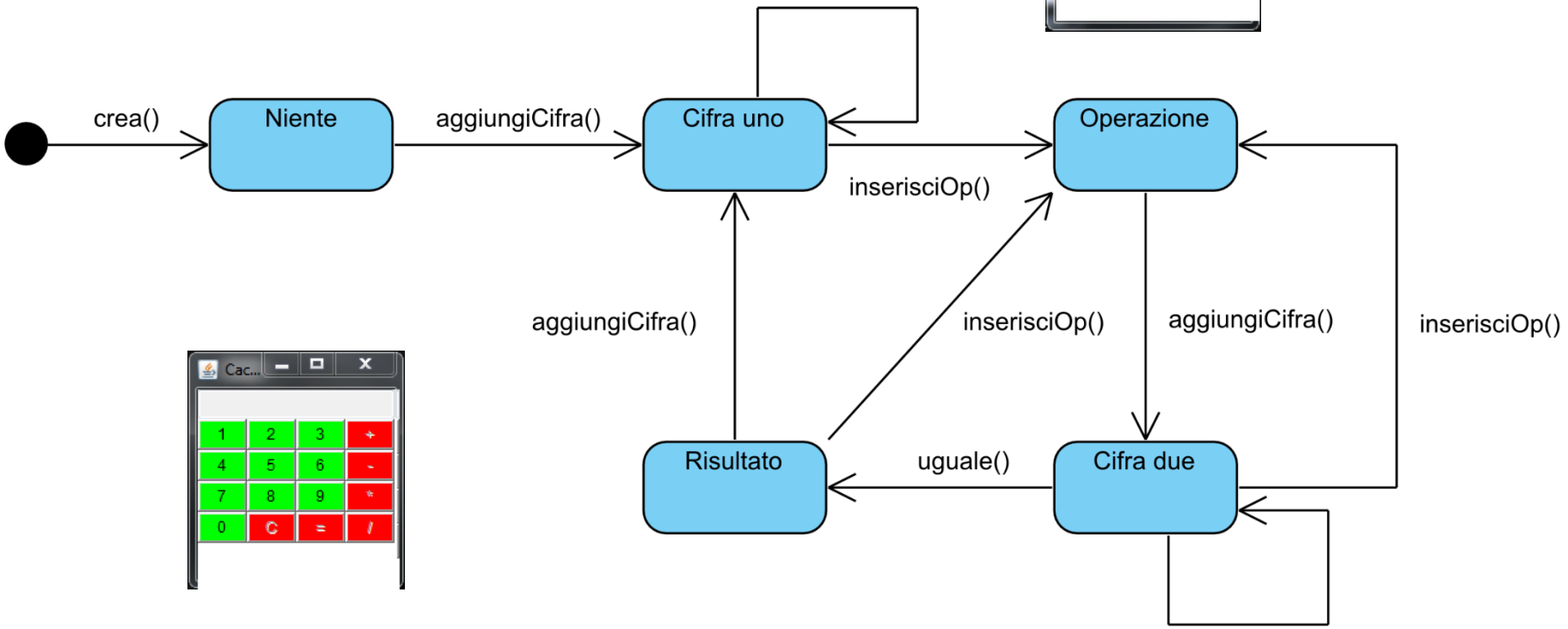
- La **classe calcolatrice** è dotata delle seguenti operazioni:
  - **crea()** per istanziare un oggetto calcolatrice
  - **aggiungiCifra()** per digitare una cifra sullo schermo
  - **inserisciOp()** per digitare un operazione
  - **uguale()** per ottenere il risultato

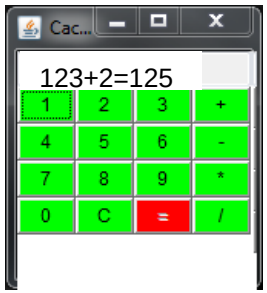
Stati di una Calcolatrice

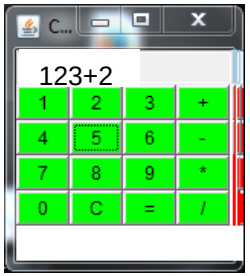
Verde -> op ammessa
Rossa -> op non ammessa

aggiungiCifra()





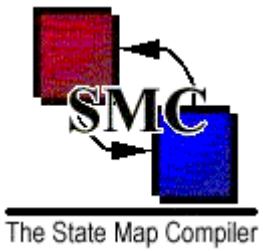




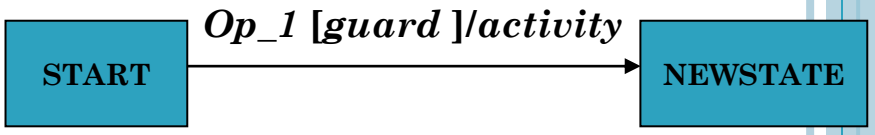
E' completa la State machine?

Come si implementa una state machine?

- Nei linguaggi che **non** le supportano direttamente
  - Es. Java
- Tre modi:
  - **Con switch**
    - Diverse varianti simili
    - Approccio + diretto (prima idea che viene in mente)
    - + laborioso da implementare e modificare
  - **Design pattern State**
    - Approccio + elegante
  - **Tabelle di stato**
    - Approccio + comodo
    - Trasformazione della tabella in codice in modo automatico
      - Utilizzo di tool "model driven"
        - SMC tool



Con Switch



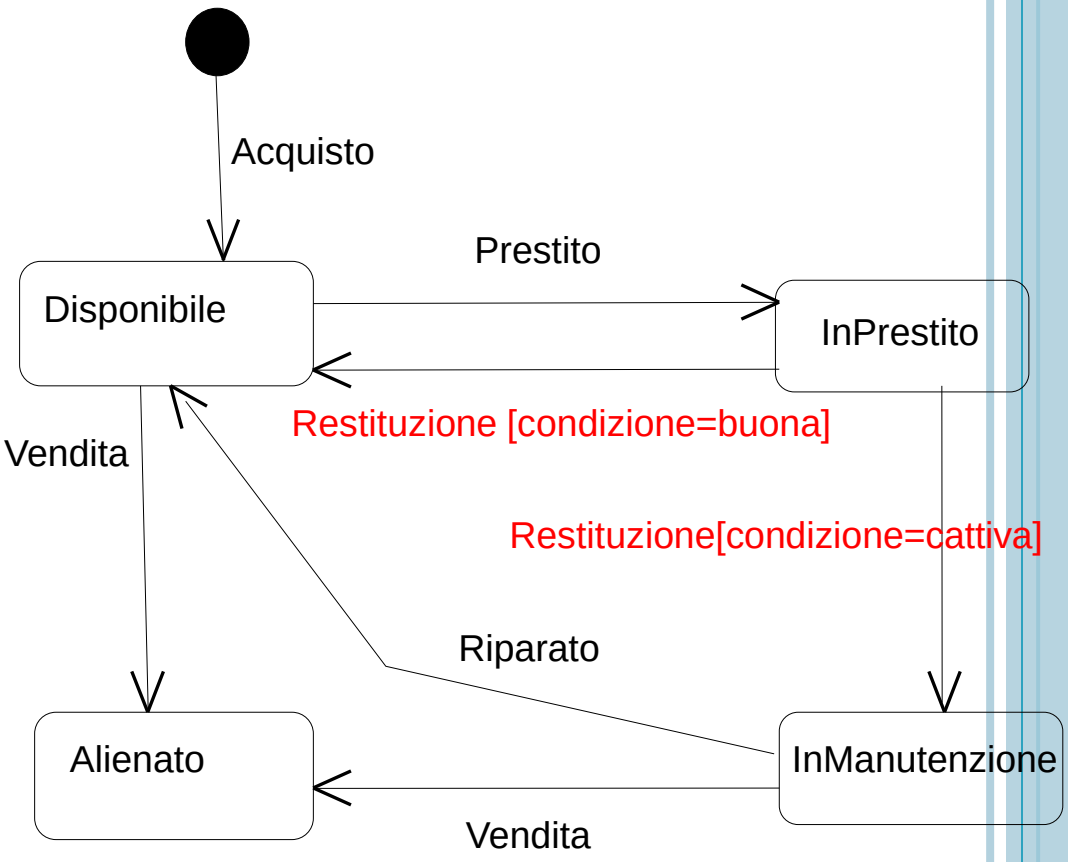
- **StateMachine MyClass**
- L'attributo ***StatoCorrente*** contiene lo stato corrente della classe MyClass
- Gli stati possono essere definiti come:
  - "private static final int"
- Costruttore
- Eventi sono rappresentati come metodi :"**Op_1, … , Op_n**"
- Le transizioni sono "case" dello switch
- case START: …
  - "source" lo stato *START*
  - "target" lo stato *NEWSTATE*
  - "evento" Op_1 (con i parametri)
  - "guardia" la condizione *guard*
  - "activity" la lista di comandi *action*

```
class MyClass {

   private int StatoCorrente;

  /*definizione stati*/
  private static final int START = 0;
  private static final int STATE_1 = 1;
  private static final int STATE_2 = 2;
  …
 /*costruttore*/
 public MyClass() { StatoCorrente=START; … }

 public void Op_1(T1 t1, … ,Tn tn) {
  switch(StatoCorrente){
     case START : { if (guard) {activity;
                          StatoCorrente=NEWSTATE;}
     case STATE_1 : {…}
     case STATE_2 : {…}
     …
   }
 }
…
```


Ci sono tre transizioni possibili associate all'evento Op_1 a secondo dello stato di partenza

Esempio copia libro

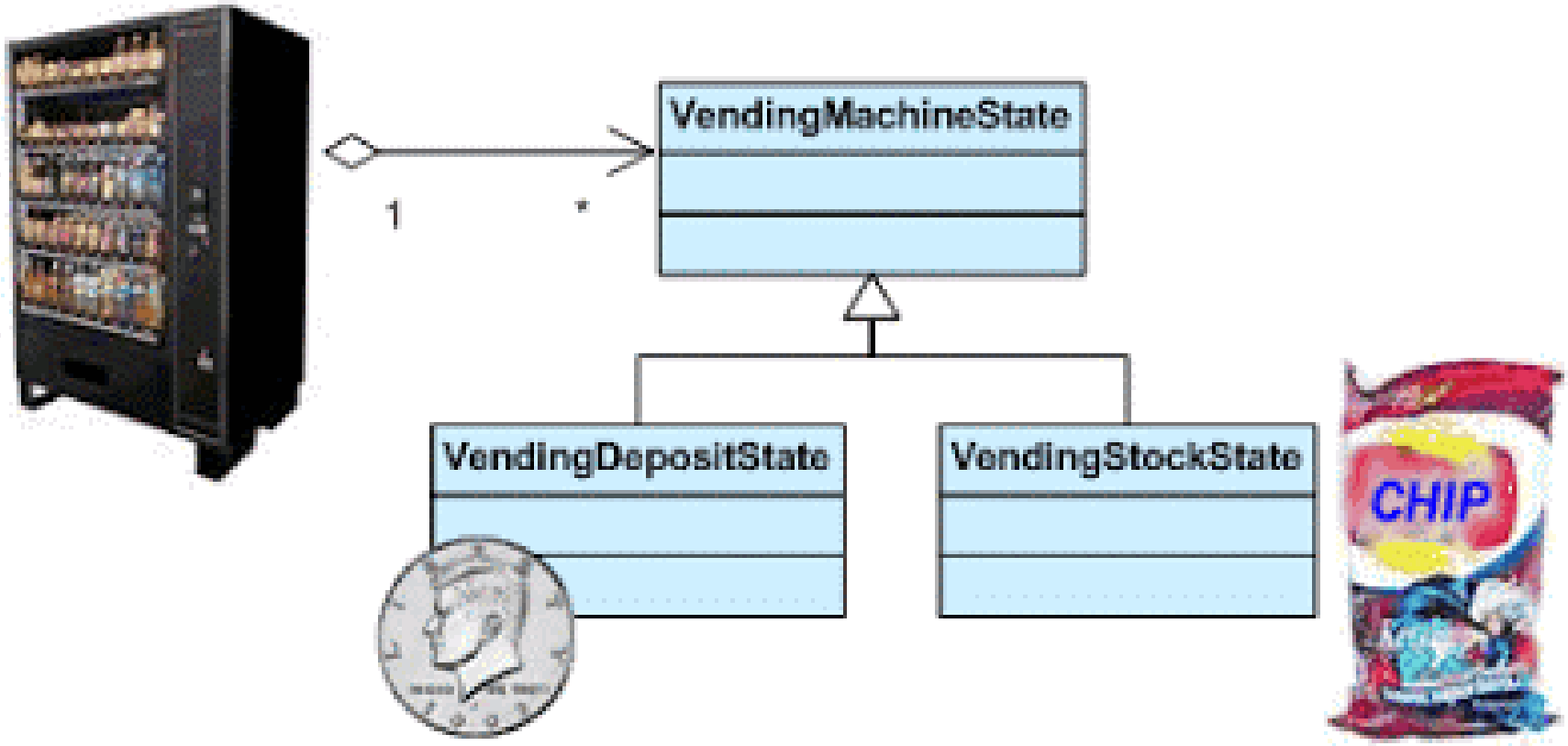
```
public class CopiaLibro {
enum condizione_copia {buona, cattiva};
private static final int DISPONIBILE=0;
private static final int INPRESTITO=1;
private static final int ALIENATO=2;
private static final int INMANUTENZIONE=3;
private int statoCorrente;
private condizione_copia condizione;
 public void Acquisto()  {
    .... Crea oggetto copia libro …
   statoCorrente=DISPONIBILE;
   condizione=condizione_copia.buona;
 }
 public void Restituzione() {
  switch(statoCorrente){
     case INPRESTITO :
       if (condizione==condizione_copia.buona) {
          …
          statoCorrente=DISPONIBILE;
        } else { // condizione della copia Cattiva
          …
         statoCorrente=INMANUTENZIONE;
     }
   default: System.out.println("Restituzione non possibile");
 }
…
}
```



Due transizioni che partono dallo stesso stato!

Design pattern State

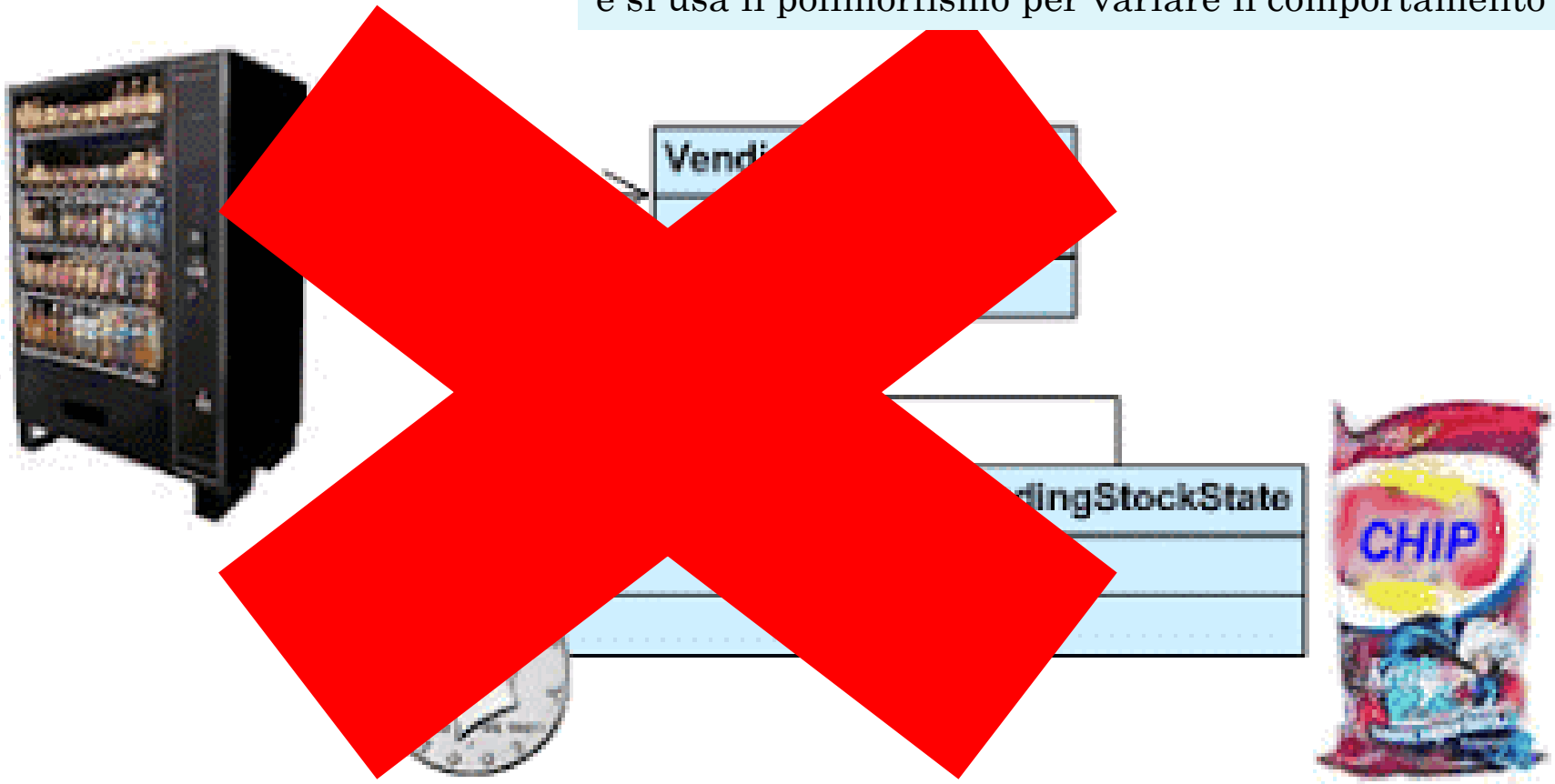
Idea di base: si rappresenta ogni stato con una classe e si usa il polimorfismo per variare il comportamento



Lo vedremo nelle lezione sui design pattern ....

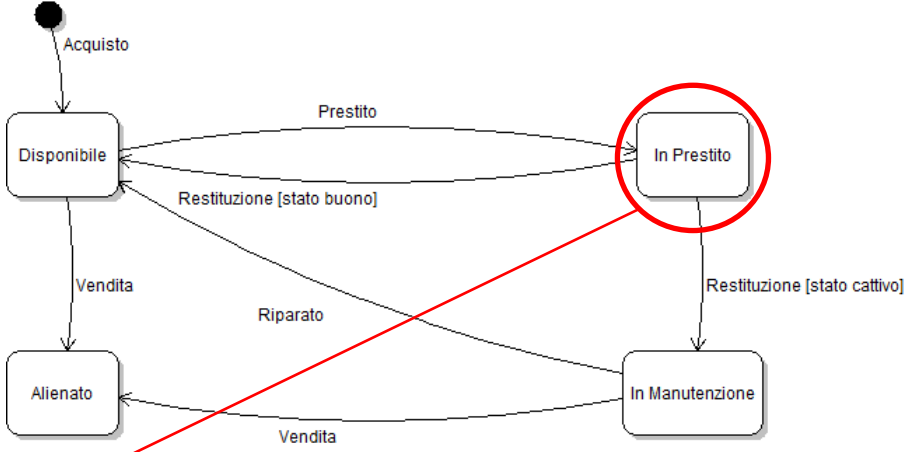
Design pattern State

Idea di base: si rappresenta ogni stato con una classe e si usa il polimorfismo per variare il comportamento



Lo vedremo nelle lezione sui design pattern ....

TABELLA DI STATO

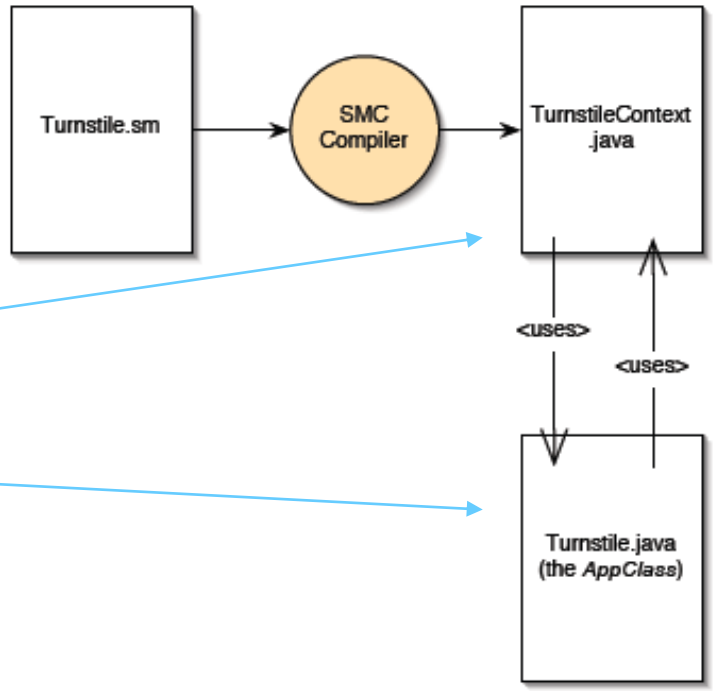


Sorgente	Destinazione	Evento	Guardia	Azione
Disponibile	InPrestito	Prestito		
InPrestito	Disponibile	Restituzione	Stato buono	
InPrestito	Manutenzione	Restituzione	Stato cattivo	
…				

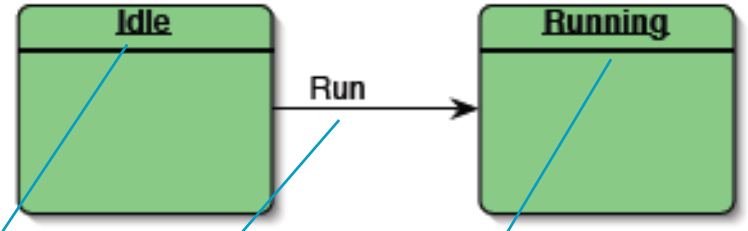
Si possono implementare direttamente in Java ma è + comodo usare tool di generazione automatica come ad esempio SMC …

USARE SMC PER GENERARE IL CODICE

1. Produrre la tabella di stato
2. Trasformarla in ".sm"
   1. Linguaggio per rappresentare la tabella
3. Fare il run del tool SMC
   - Produce "file" che contiene la logica della state machine
   - Codice che segue "state design pattern"
4. Implementare AppClass
   - Contiene l'implementazione delle attività ed espone l'interfaccia degli eventi
5. Interagire con AppClass "generando eventi"
   - Visti come metodi in AppClass



FORMATO .SM



Transition | Next State | Actions

```
/* smc recognizes c-style comments */
// .as well as c++ style comments

Idle
{
    Run         Running         {}
}
```

49

TORNELLO

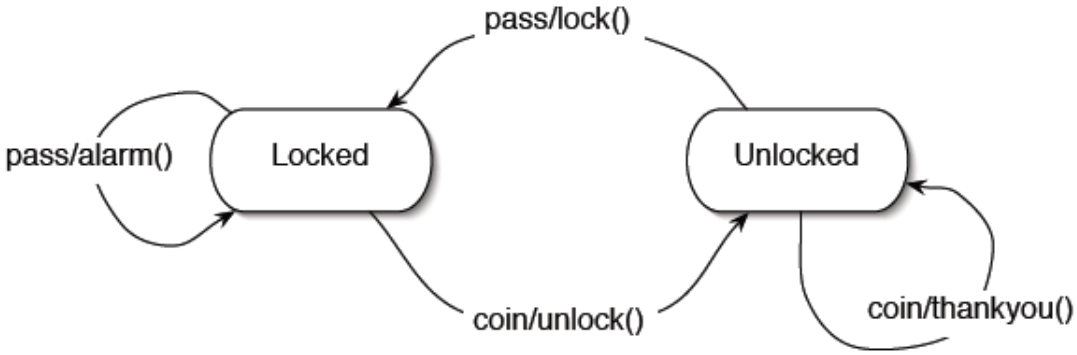


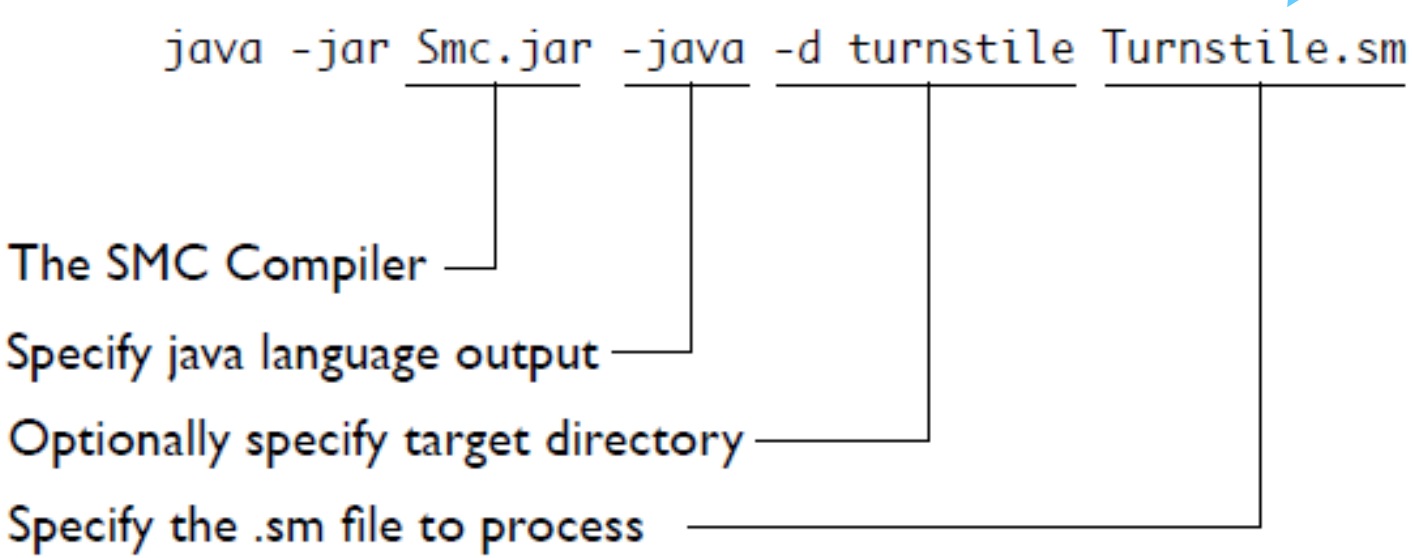
Tabella di stato

State	*Event*	Next State	Action
Locked	*coin*	Unlocked	unlock
	pass	Locked	alarm
Unlocked	*coin*	Unlocked	thankyou
	pass	Locked	lock

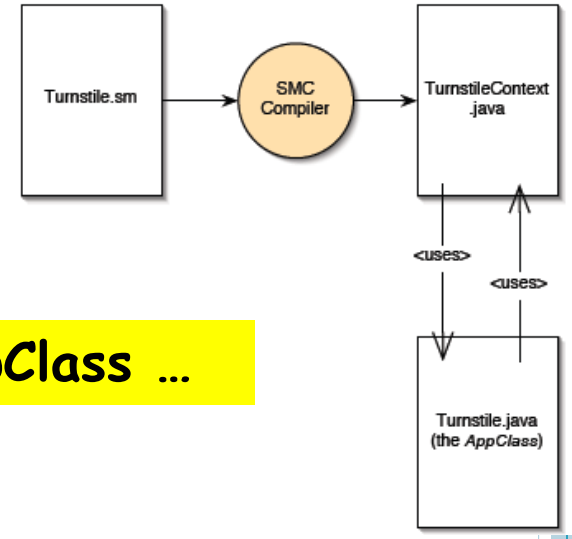
http://smc.sourceforge.net/

RUN DEL TOOL SMC

Tabella di stato



AppClass (Turnstile.java)



Lo sviluppatore deve scrivere l'AppClass …

```
 9 public class Turnstile
10 {
11     TurnstileContext _fsm;
12
13
14     public Turnstile(TurnstileActions actions)
15     {
16         _fsm = new TurnstileContext(this);
17
18     }
19
20     public void coin() { _fsm.coin(); }
21     public void pass() { _fsm.pass(); }
22
23     public void alarm() {                    }
24     public void lock() {                  }
25     public void thankyou() {                 }
26     public void unlock() {                   }
27
```

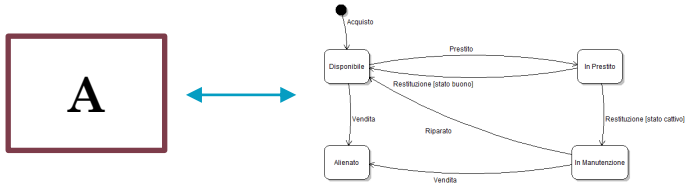
Define and instantiate "fsm" context class (generated)

Expose transition calls to other parts of application, if necessary

Implement (or delegate) actions

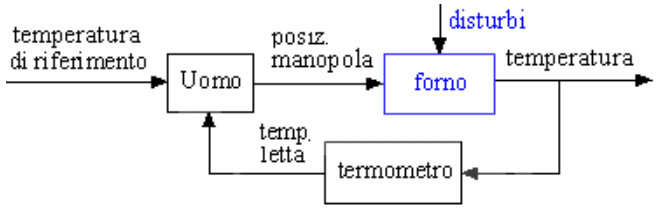
RICAPITOLANDO ….

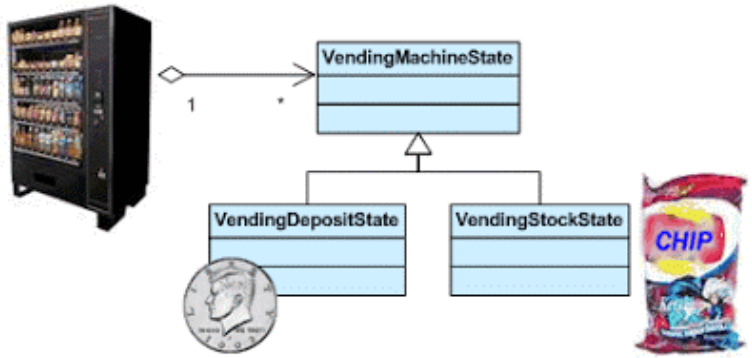
COMPORTAMENTO



Sintassi e semantica

Dove sono usate?



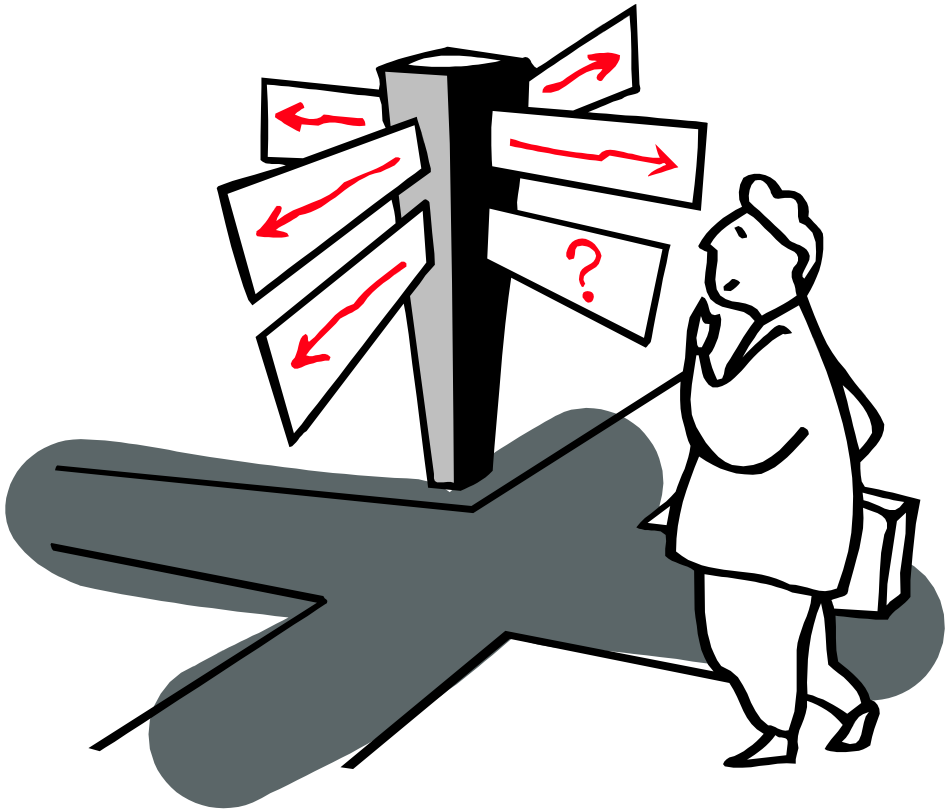


Implementazione

Materiale e riferimenti

- Per realizzare la seguente presentazione sono stati utilizzati:
  - UML distilled M. Fowler
  - Usare UML. Ingegneria del Software con oggetti e componenti P. Stevens e R. Pooley
  - Slide di Guglielmo De Angelis (Istituto Faedo)
  - Slide di Angelo di Iorio (UniBo) AA. 2010-2011
  - Documentazione tool SMC

The end ...



Domande?