UML State Diagrams

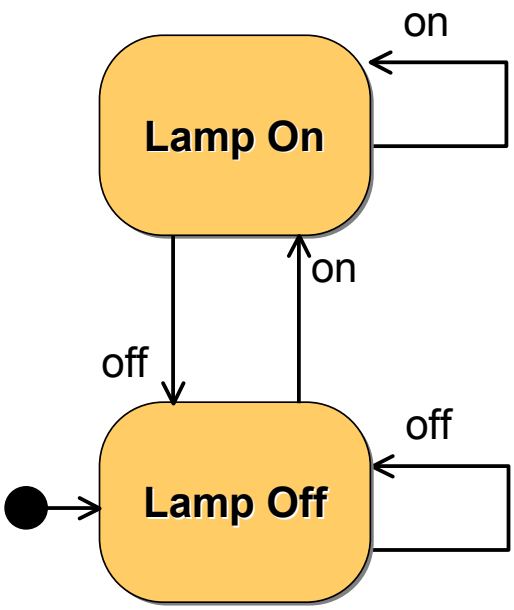
A. Fantechi

1/4/2010

Macchina a Stati Finiti

(Automa a stati finiti, Finite State Machine, FSM)

- Descrizione grafica del comportamento di una FSM



Macchina a Stati Finiti

(Automa a stati finiti, Finite State Machine, FSM)

- Descrizione formale del comportamento di una FSM:

- Una FSM *M* è una quadrupla (S, s_0, E, R) dove:
  - S è un insieme finito di stati
  - $s_0 \in S$ è lo stato iniziale
  - E è un insieme di eventi (*trigger*)
  - $R \subseteq S \times E \times S$ è la relazione di transizione.

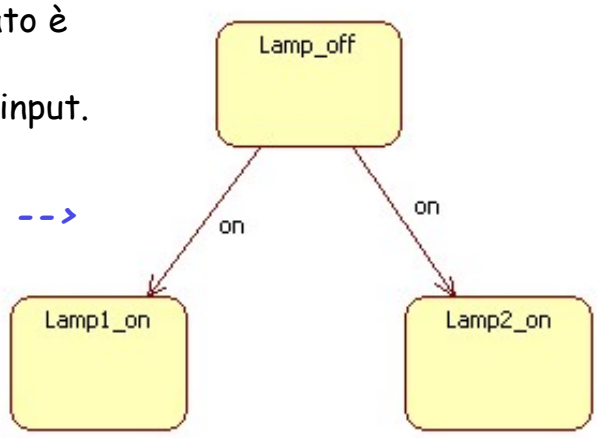
    $(s_1, e, s_2) \in R$ si scrive anche $s_1 \xrightarrow{e} s_2$

Nondeterminismo

- Spesso si usa una *funzione di transizione* invece di una relazione di transizione:
  - $R: S \times E \rightarrow S$

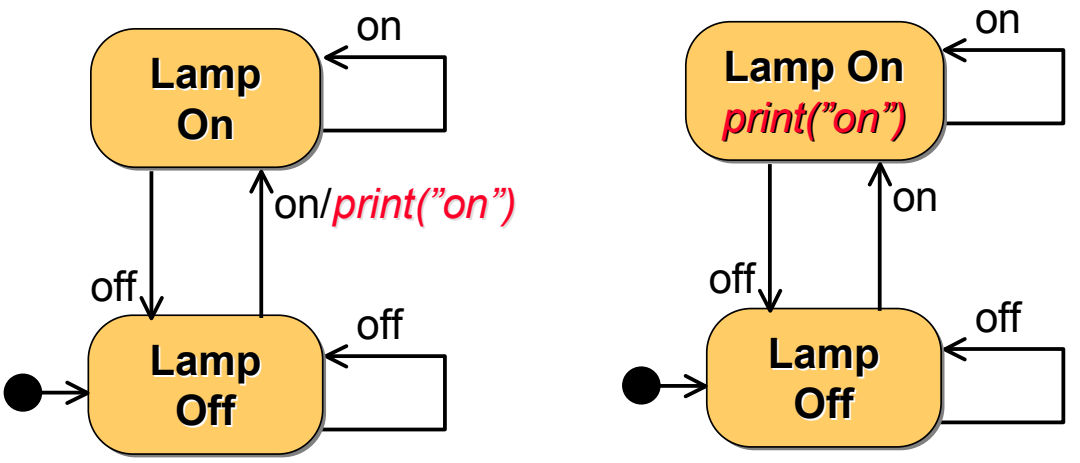
  In questo modo il prossimo stato è sempre determinato univocamente dall'evento di input.

Caso di nondeterminismo --->



Uscite, azioni

- Uscite generate dall'automa:



Descrizione formale macchine Mealy e Moore

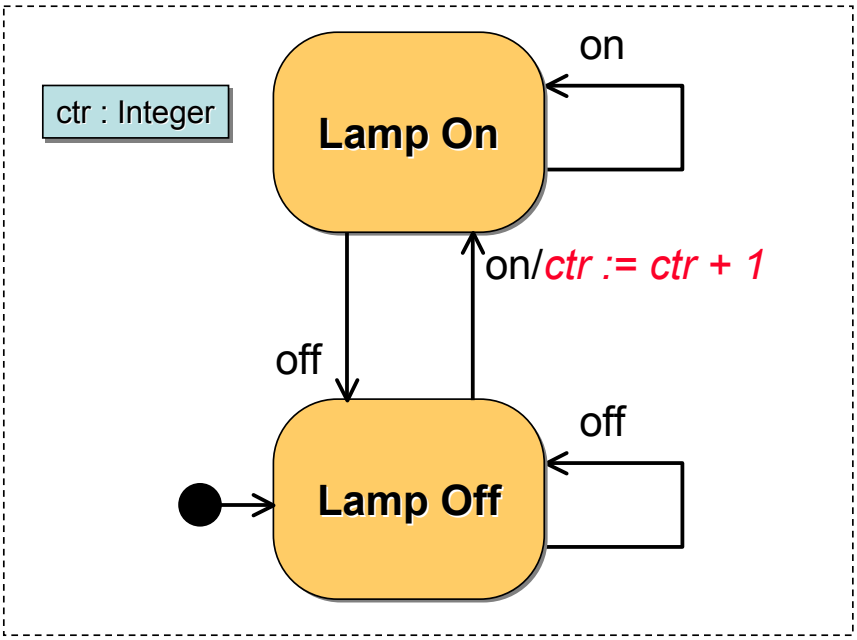
- Una Macchina di Mealy è una sestupla (S, s_0, E, R, O, o) dove:
  - S è un insieme finito di stati
  - $s_0 \in S$ è lo stato iniziale
  - E è un insieme di eventi (*trigger*)
  - $R : S \times E \rightarrow S$ è la funzione di transizione.
  - O è un insieme di possibili uscite
  - $o : S \times E \rightarrow O$ è la funzione di uscita

- Una Macchina di Moore è una sestupla (S, s_0, E, R, O, o) dove:
  - S è un insieme finito di stati
  - $s_0 \in S$ è lo stato iniziale
  - E è un insieme di eventi (*trigger*)
  - $R : S \times E \rightarrow S$ è la funzione di transizione.
  - O è un insieme di possibili uscite
  - $o : S \rightarrow O$ è la funzione di uscita

(determinismo)

Extended Finite State Machines (EFSM)

- Estensione con variabili ("extended state")



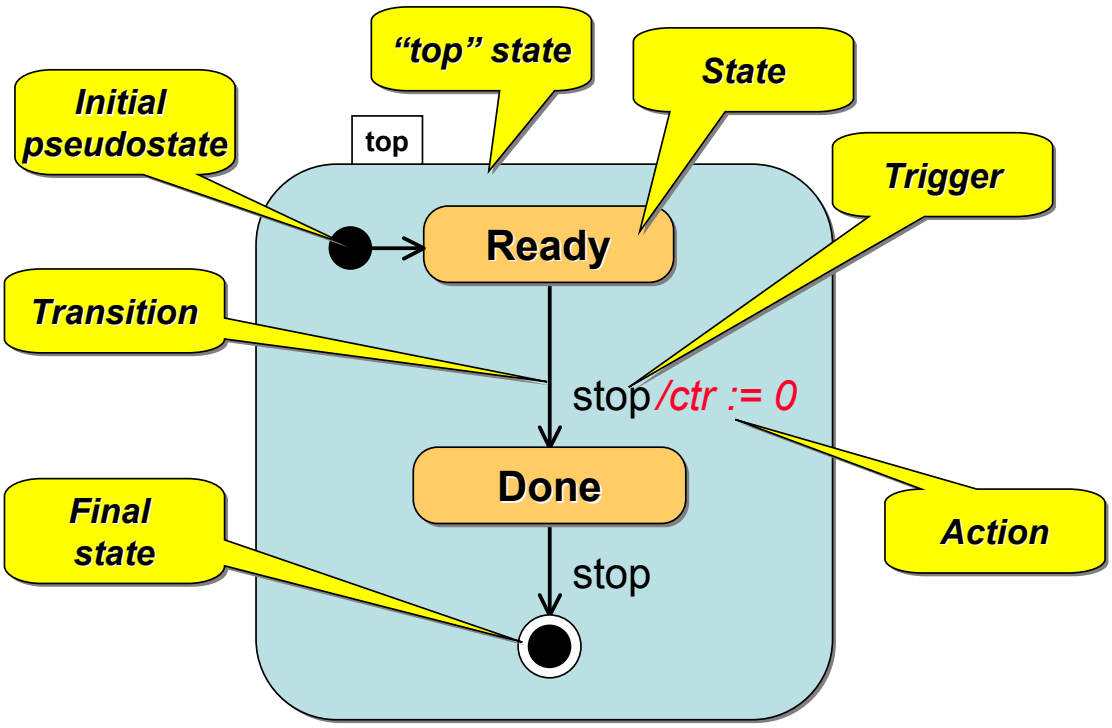
EFSM

- Una EFSM (di Mealy) è definita da:
  - Un insieme di segnali di input (input alphabet)
  - Un insieme di segnali di output (output alphabet)
  - Un insieme di stati
  - Un insieme di transizioni
    - segnale di trigger
    - guardia
    - azione
  - Un insieme di variabili (extended state variables)
  - Un'indicazione di stato iniziale (stato iniziale + valore iniziale delle variabili)
  - Un insieme di stati finali (vuoto nel caso di macchina non terminante)

Harel Statecharts

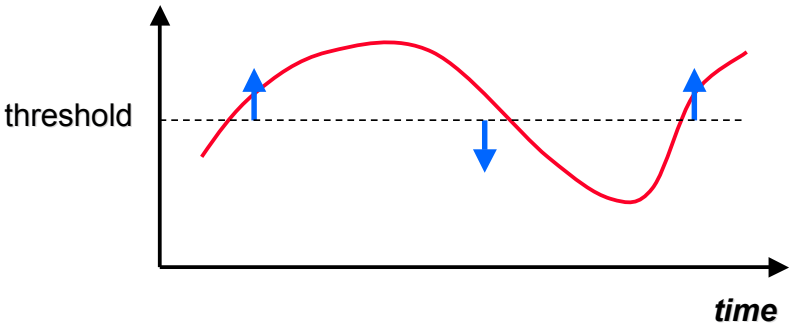
- Create da David Harel (I-Logix) alla fine degli anni '80, come ulteriore estensione delle EFSM
- Formano la base del modello comportamentale di UML
- Supportano
  - **Stati annidati (gerarchie di stati)**
  - Azioni su
    - Transizioni
    - Entry (ingresso in uno stato)
    - Exit (uscita da uno stato)
  - Attività eseguite all'interno di uno stato (Do)
  - Guardie
  - History
  - Eventi di Broadcast
  - Regioni Ortogonali (AND-States)

Basic UML Statechart Diagram



Che tipo di comportamento?

- In generale, le EFSM sono adatte per descrivere un comportamento dicreto, event-driven
  - inappropriate per modellare comportamento continuo
  - in UML non ci sono strumenti per modellare un comportamento continuo



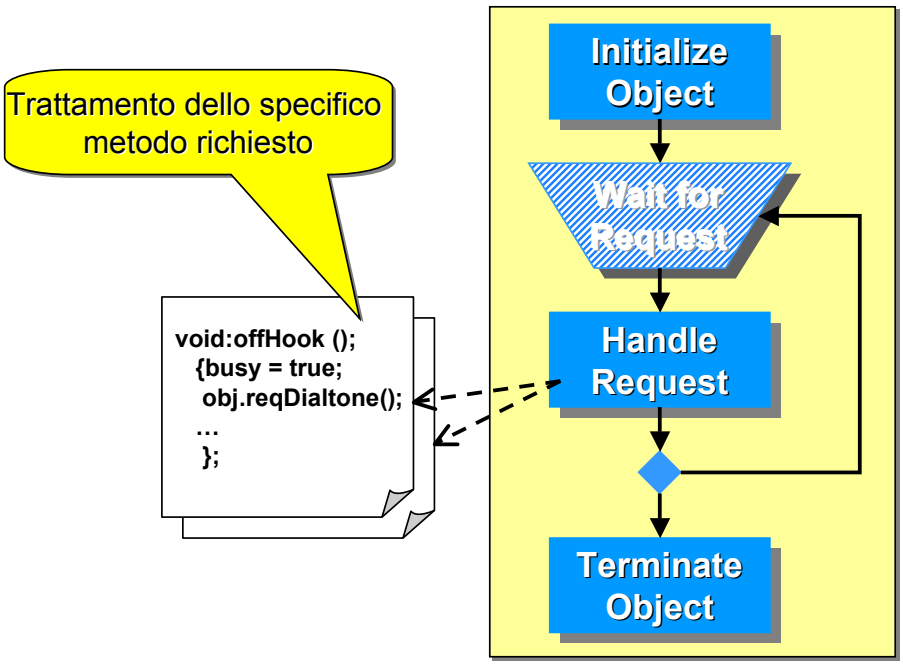
Comportamento Event-Driven

- Evento = un tipo di occorrenza di qualcosa di osservabile
  - interazioni:
    - Invocazione sincrona di un metodo di un oggetto (*call event*)
    - Ricezione asincorna di un segnale (*signal event*)
  - Occorrenza di istanti di tempo (*time event*)
    - Scadenza di un intervallo
    - Tempo di clock o di calendario
  - Aggiornamento del vlore di una variabile (*change event*)
- Istanza di un Evento (Event Instance) = un'istanza di un tipo di evento che accade in un certo istante di tempo e che non ha durata

Comportamento Event-Driven - UML

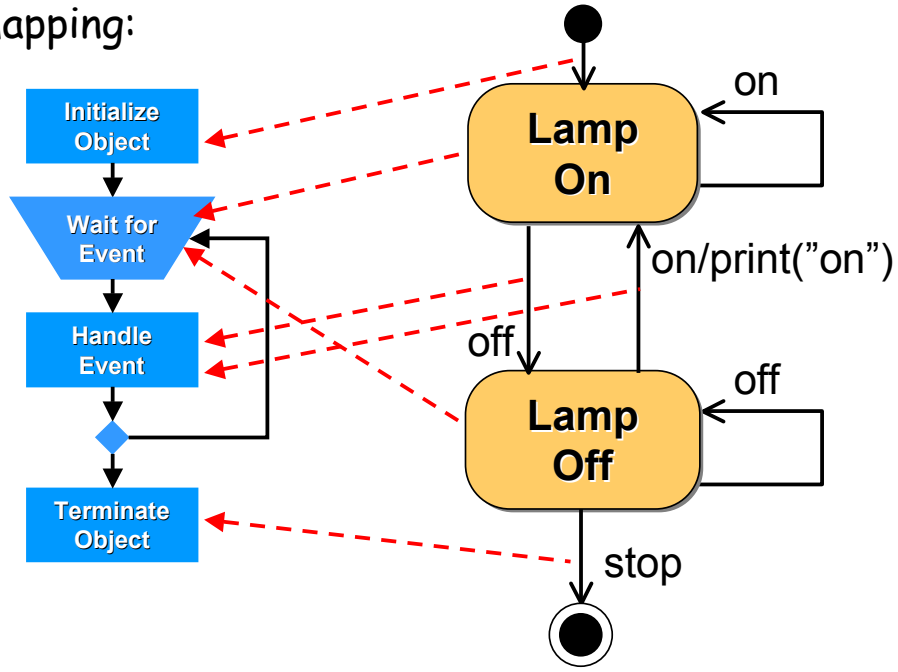
- In linea di principio, si può parlare di comportamento event-driven per un qualsiasi tipo di sistema
- In UML, il comportamento event-driven è quello manifestato dagli oggetti. Quindi possiamo associare agli oggetti di una classe un comportamento event-driven specificato attraverso uno state diagram.
- La semantica degli state diagram UML è data principalmente in relazione ad **oggetti *attivi***

Comportamento di un oggetto Passivo - Modello Generale



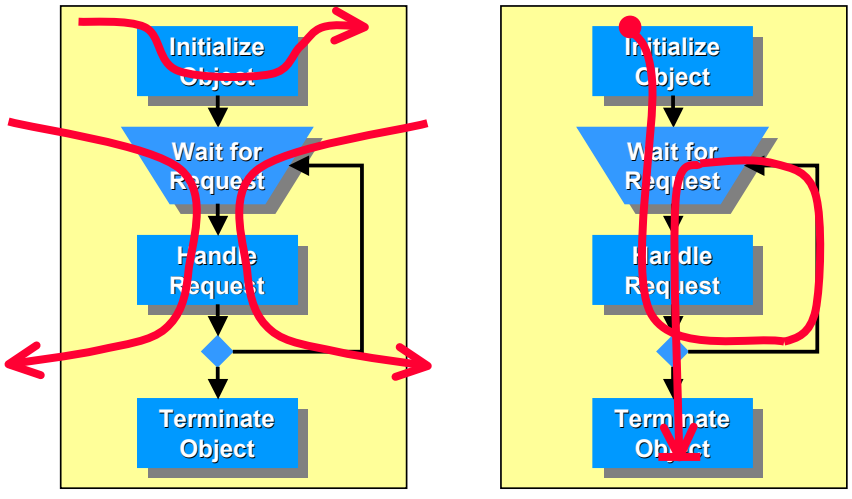
Oggetti Passivi e Macchine a Stati

- mapping:

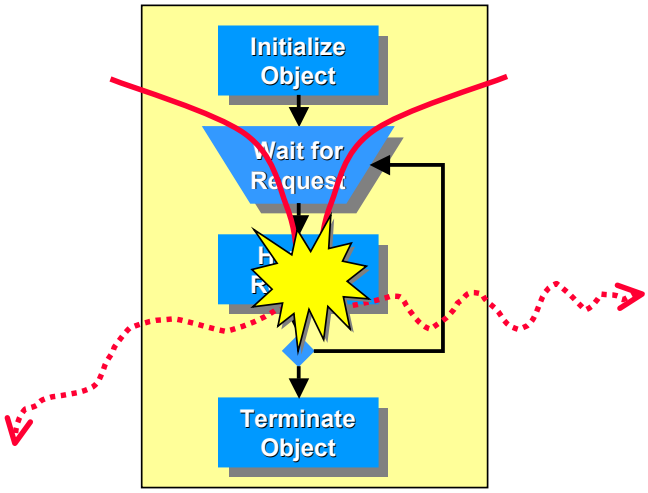


Oggetti e flussi di controllo (threads)

- **Oggetti Passivi:** flusso di controllo esterno
- **Oggetti Attivi:** proprio flusso di controllo



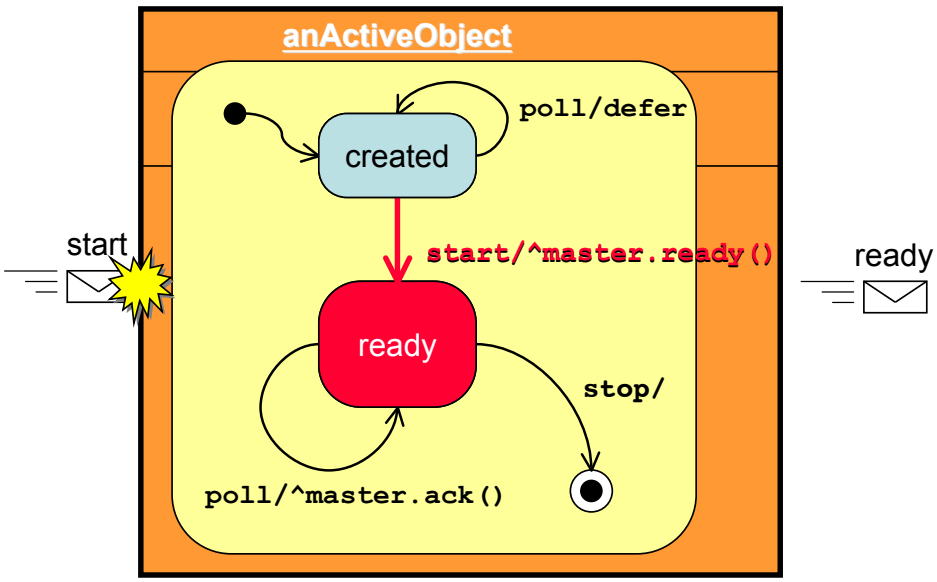
Flussi concorrenti



- Un oggetto passivo può essere attraversato da flussi concorrenti:
  --> *problema di sincronizzazione*
- Un oggetto attivo ha un proprio, unico, flusso: la concorrenza si ottiene con due oggetti attivi, istanze distinte della stessa classe

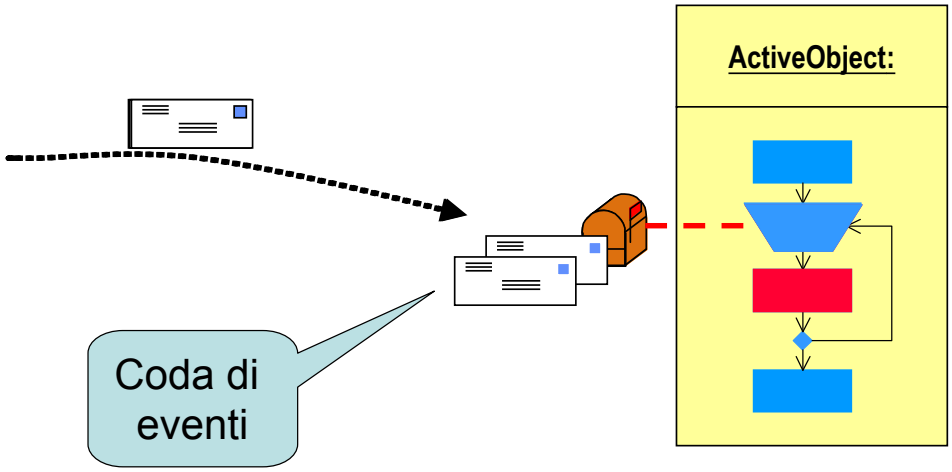
Oggetti Attivi e Macchine a Stati

- Oggetti Attivi con il proprio flusso di controllo



Le variabili di stato esteso sono gli attributi dell'oggetto.

Semantica degli Oggetti Attivi

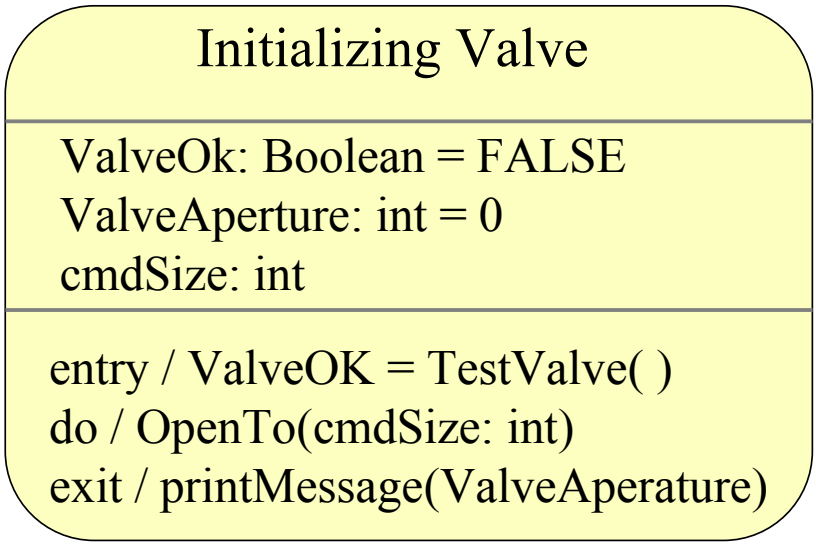


Run-to-completion model

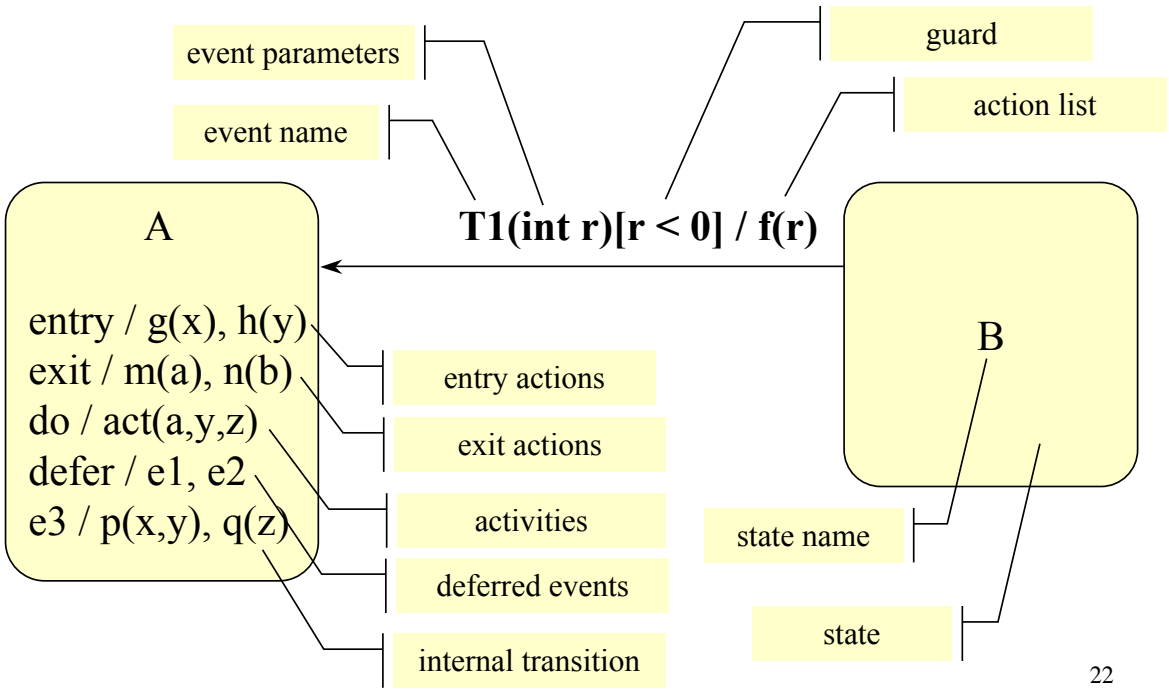
Run-to-completion model

- Il run-to-completion model step definisce come avviene una transzione tra due configurazioni della macchina a stati.
- *Configurazione = stato corrente + valore delle variabili di stato esteso + situazione della coda in ingresso*
- Un evento in coda (*received*) può essere prelevato (*dispatched*) dalla coda e trattato solo quando l'elaborazione dell'evento precedente è terminata (*consumed*).
- Durante uno step di run-to-completion può essere eseguita una sequenza di varie attività, che possono includere la modifica di un attributo locale, l'invio di un segnale, l'invocazione di un metodo di un altro oggetto, la rimozione di un evento dalla coda di ingresso

Stati UML



Sintassi completa



Eventi

- Un *evento* può essere:
  - Una condizione che diventa vera (*guardia*)
  - Ricezione di un esplicito segnale da un altro oggetto
  - Ricezione di una chiamata a un metodo da un altro oggetto
  - Timeout (passage of a specified interval)
- Eventi possono causare *transizioni* di *stato*

(etichette sulle) Transizioni

event-name (parameter list) [guard] / action expression

- ***event-name*** nome dell'evento di trigger per la transizione
- ***parameter list*** parametri con tipo (opzionale)
- ***guard*** espressione Booleana sui parametri e sugli attributi dell'oggetto (variabili)
- ***action-expression*** lista di operazioni da eseguire quando la transizione viene selezionata: la lista è eseguita ***atomicamente***

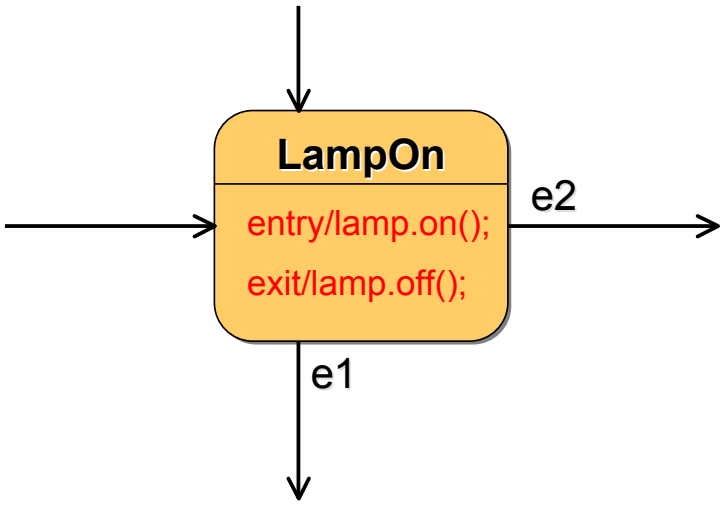
Esempi di etichette sulle transizioni

- JustDoIt
- JustDoIt(x)
- JustDoIt(x: int)
- JustDoIt(x) [x>0]
- JustDoIt [y<10] / print(y)
- JustDoIt(x,y) [x>y+10] / print (x+y); target->genSignal(Nike.MakeMoney(MuchoDollaro))

Tipi di Eventi

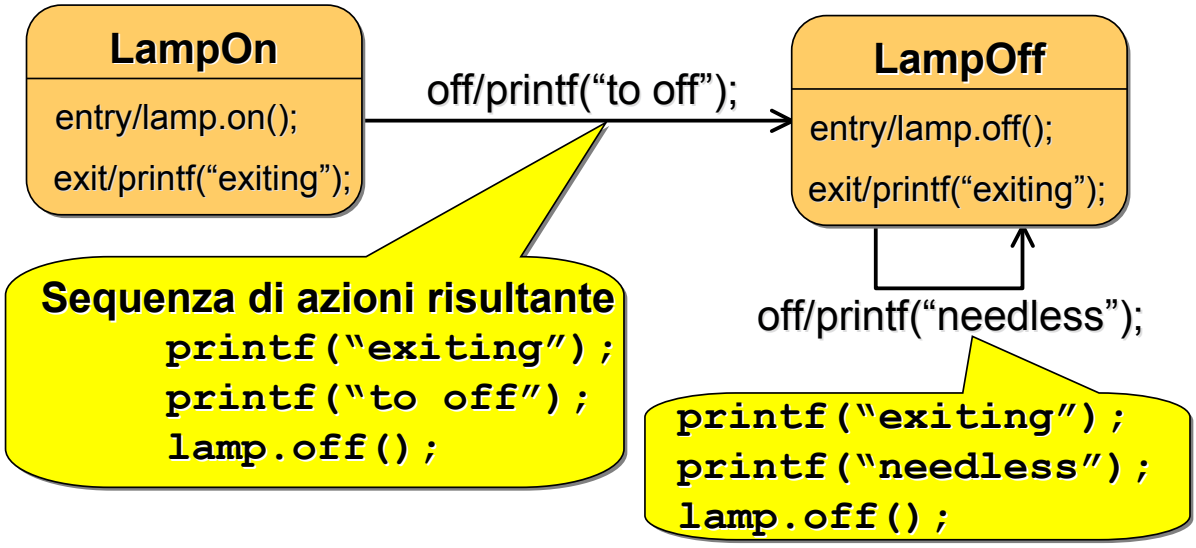
- UML definisce 4 tipi di eventi
  - Signal Event
    - Ricezione di un segnale Asinchrono
    - e.g. evFlameOn
  - Call Event
    - Ricezione di un'invocazione di un metodo
    - e.g. op(a,b,c)
  - Change Event
    - Aggiornamento del valore di una variabile (condivisa)
  - Time Event
    - Scadenza di un tempo Relativo (intervallo)
    - Raggiungimento di un tempo Assolute
    - e.g. tm(PulseWidthTime)

State Entry and Exit Actions



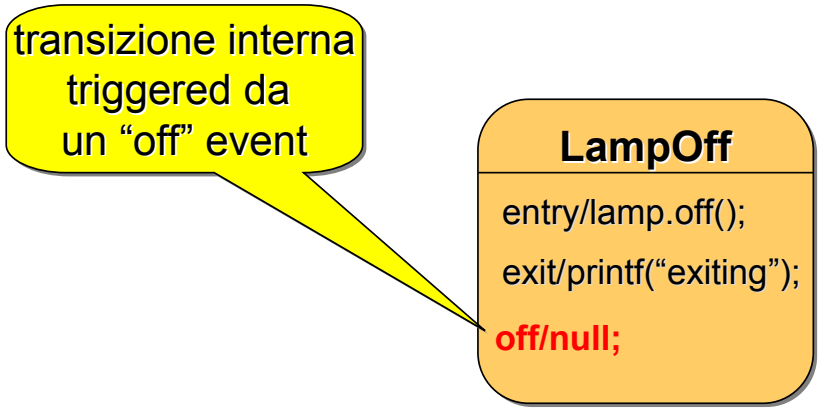
Ordine delle Azioni: un caso semplice

- Exit actions precedono le transizioni
- Entry action seguono le transizioni



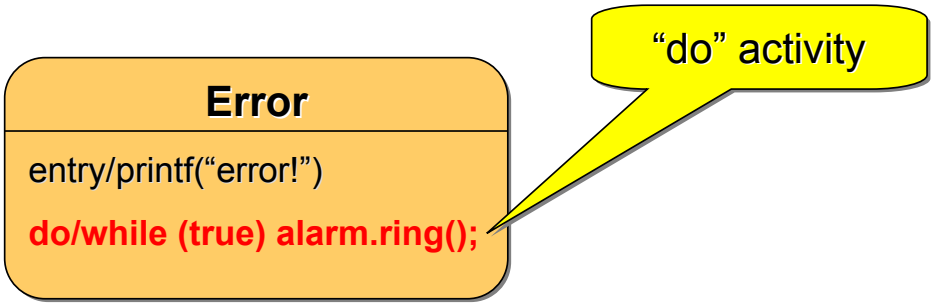
Transizioni Interne

- Self-transitions
- Non provocano l'esecuzione di entry e exit actions



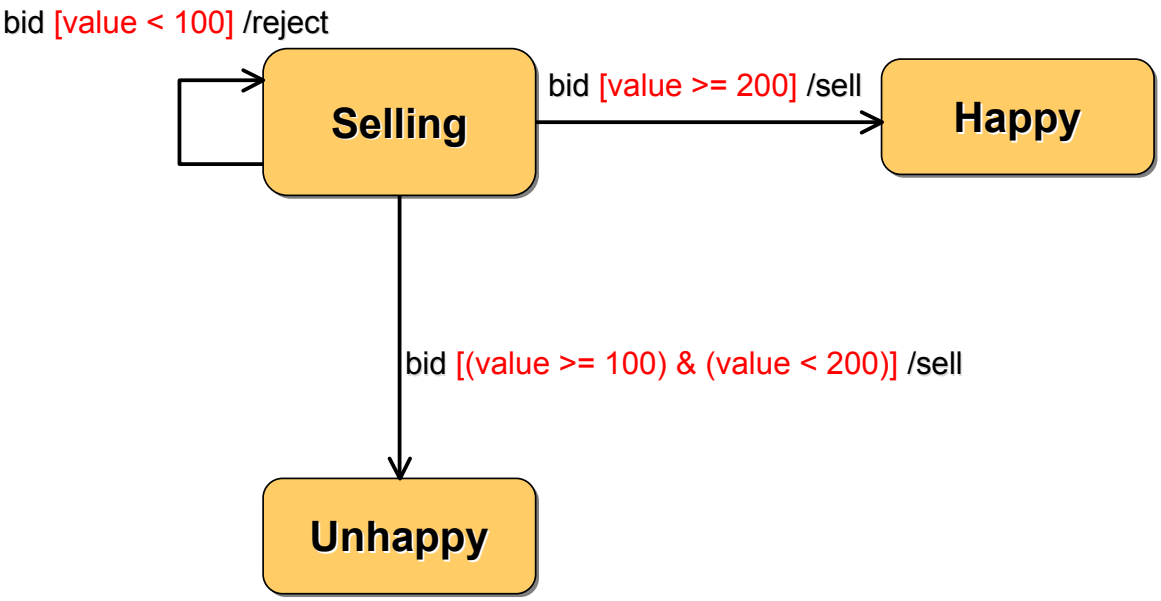
State ("Do") Activities

- thread concorrente che esegue fino a che:
  - l'attività è completata *oppure*
  - si esce dallo stato con una transizione uscente



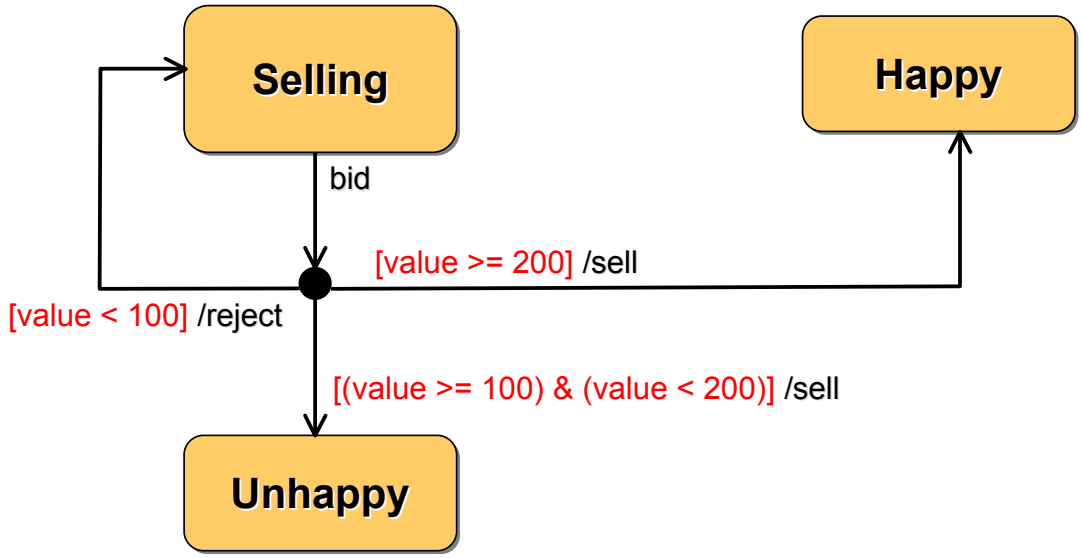
Guardie

- Esecuzione Condizionale di transizioni
  - Le guardie (predicati Booleani) non devono avere side-effect



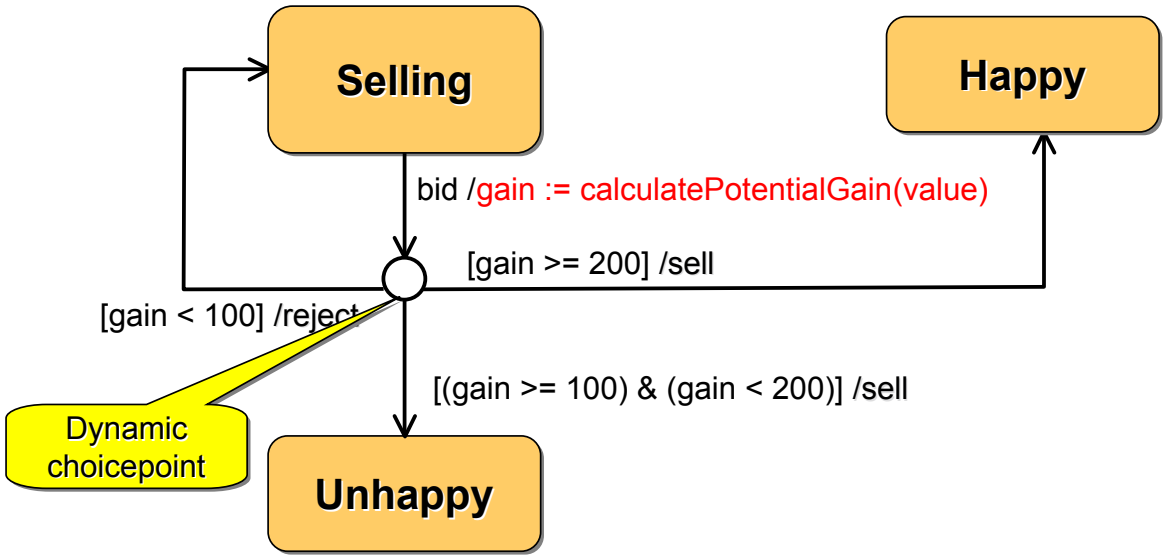
Static Conditional Branching

- Abbreviazione grafica per evidenziare una signola transizione uscente



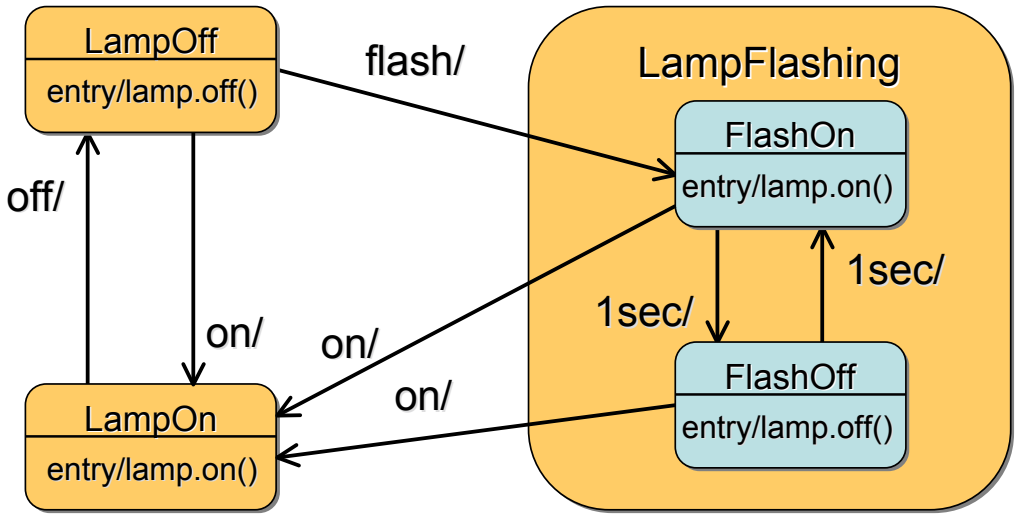
Dynamic Conditional Branching

- ***Choice pseudostate***: le guardie vengono valutate solo quando viene raggiunto lo pseudostato.



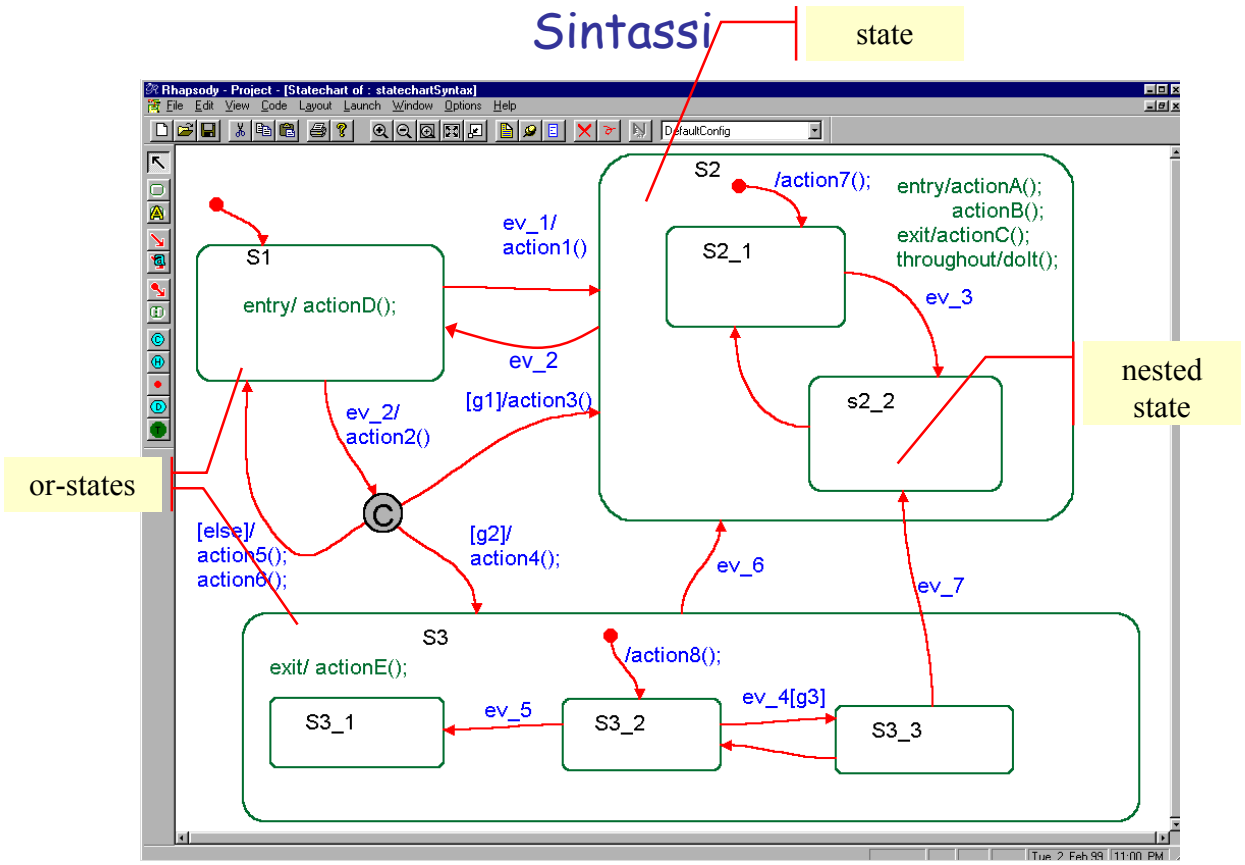
Macchine a stati gerarchiche

- stati decomposti ricorsivamente in macchine a stati

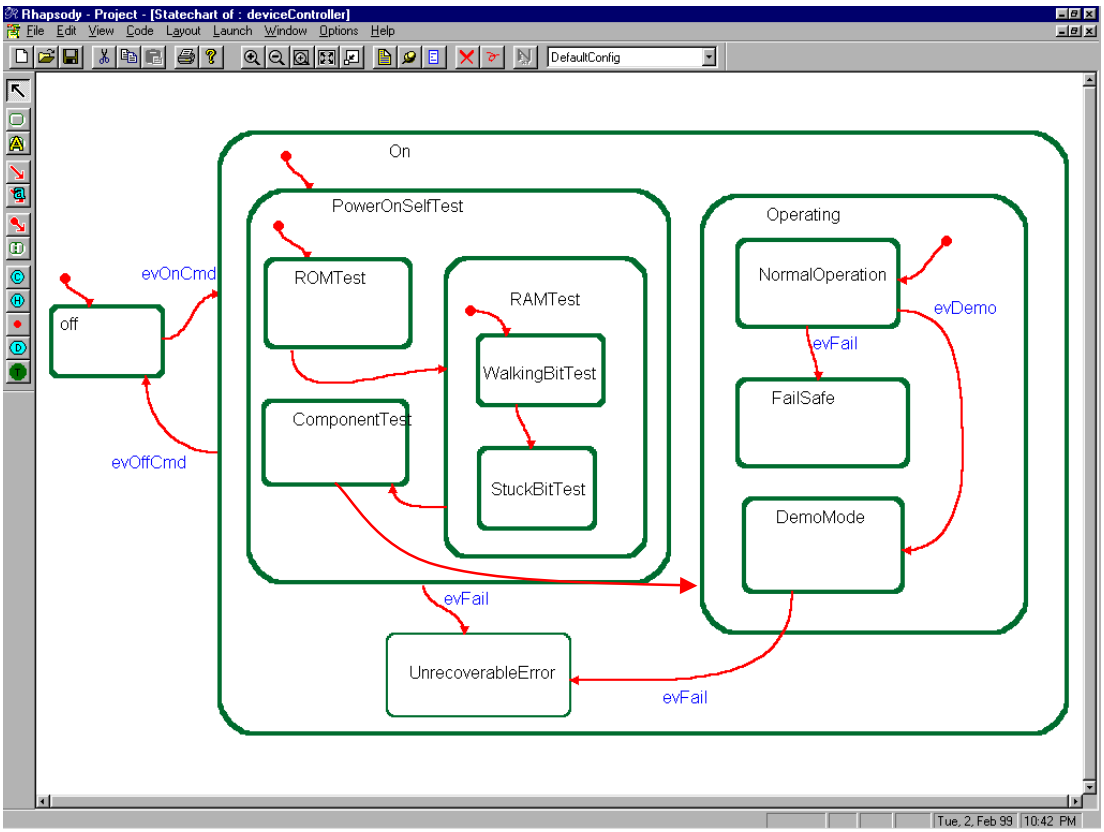


OR States

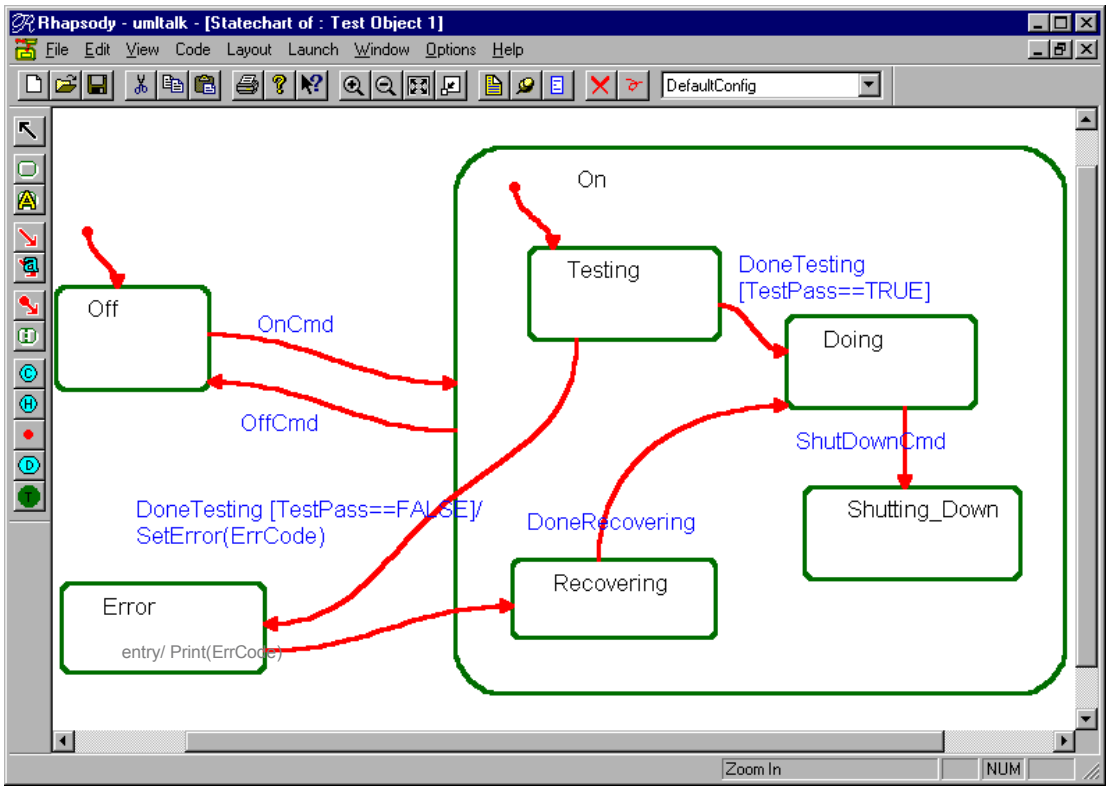
- Migliorano la scalabilità
- Migliorano la comprensibilità
- Permettono la decomposizione gerarchica di un problema (*divide-and conquer*)
- Possibilità:
  - Stati annidati sullo stesso diagramma
  - Stati annidati su diagrammi separati
    - aka ***"submachines"***



OR States

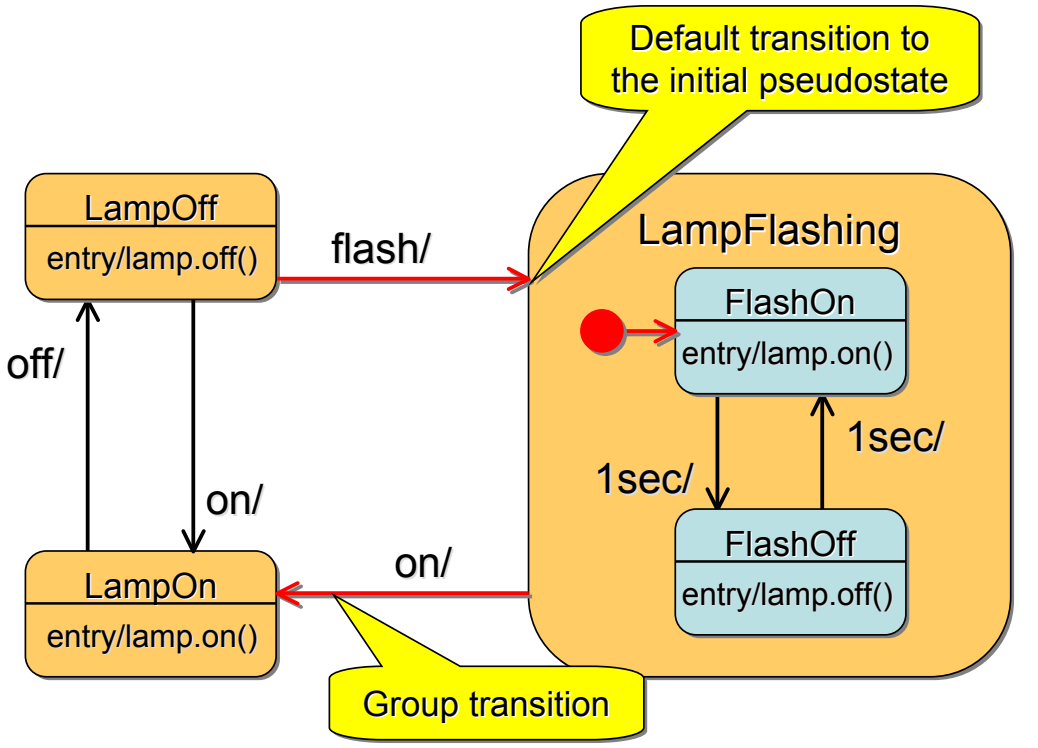


Statechart gerarchica



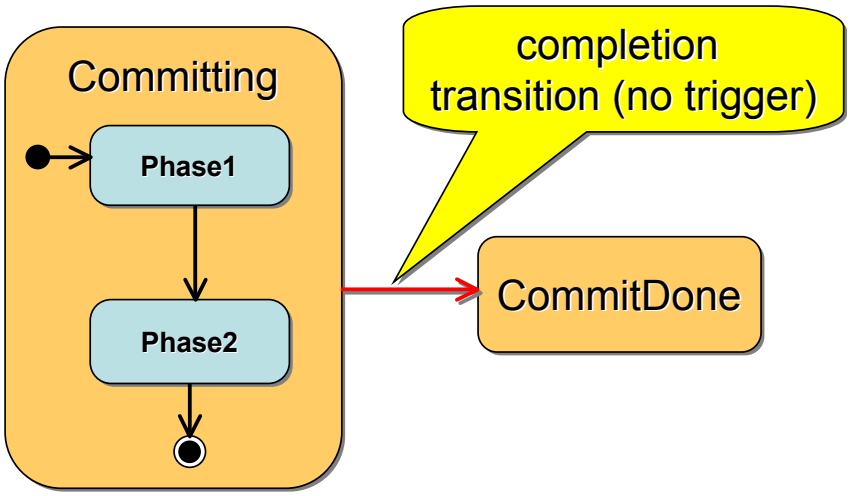
Group Transitions

- Transizioni a più alto livello nella gerarchia di stati



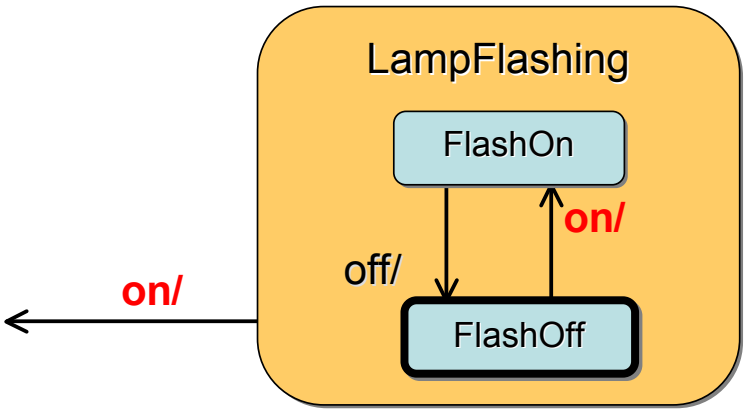
Completion Transitions

- Triggered da un evento di *completion*
  - generato automaticamente quando uno macchina immediatamente annidata termina



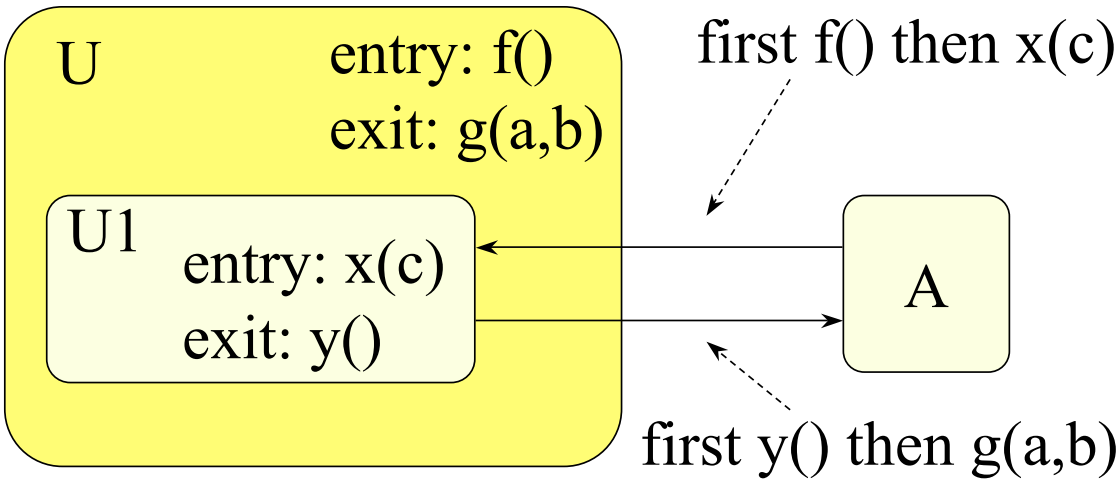
Regole di Triggering

- Due o più transizioni possono avere lo stesso evento trigger
  - La transizione più interna ha la precedenza
  - Un evento in coda è consumato sia che attivi una transizione sia che non attivi alcuna transizione.



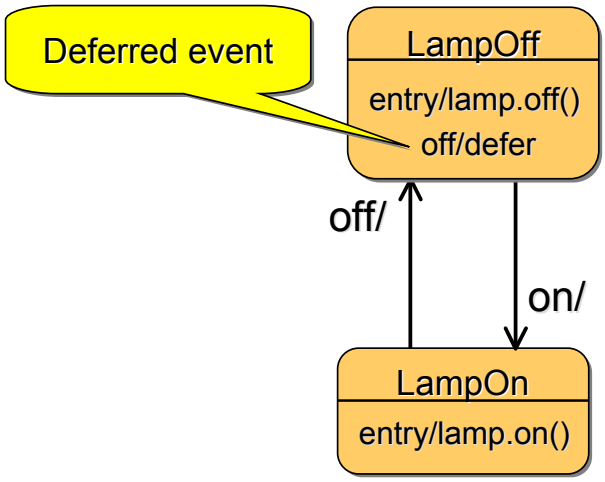
Ordine delle azioni annidate

- Da fuori a dentro le azioni entry
- Da dentro a fuori le azioni exit

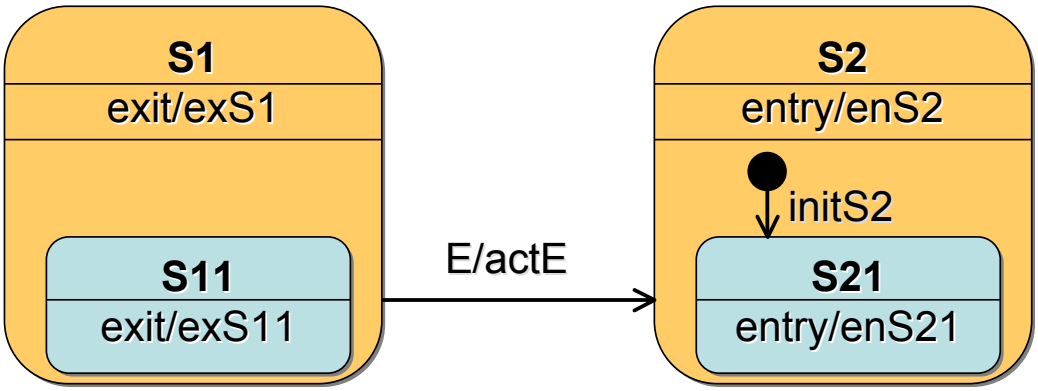


Deferred Events

- Gli eventi in coda che non possono attivare una transizione vengono scartati, fino a che un evento in grado di attivare una transizione non venga trovato nella coda.
- ***Deferred events***
  - Se però un evento è in una "deferred" list, non viene scartato e rimane in coda (finchè rimane nella deferred list)
  - Se l'oggetto transisce in uno stato dove l'evento non è più presente in una deferred list, tale evento viene scartato.



Ordine delle Azioni: Caso Complesso



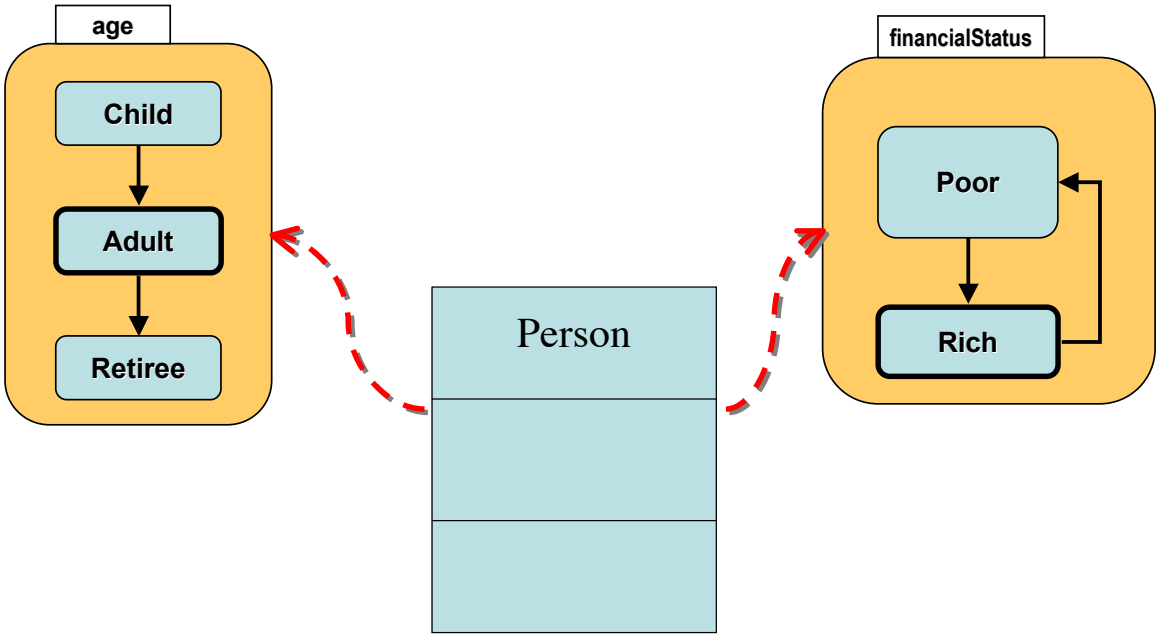
sequenza di esecuzione delle Azioni,

in caso di triggering della transizione

exS11 ⇨ exS1 ⇨ actE ⇨ enS2 ⇨ initS2 ⇨ enS21

Ortogonalità

- Viste multiple simultanee sulla stessa entità



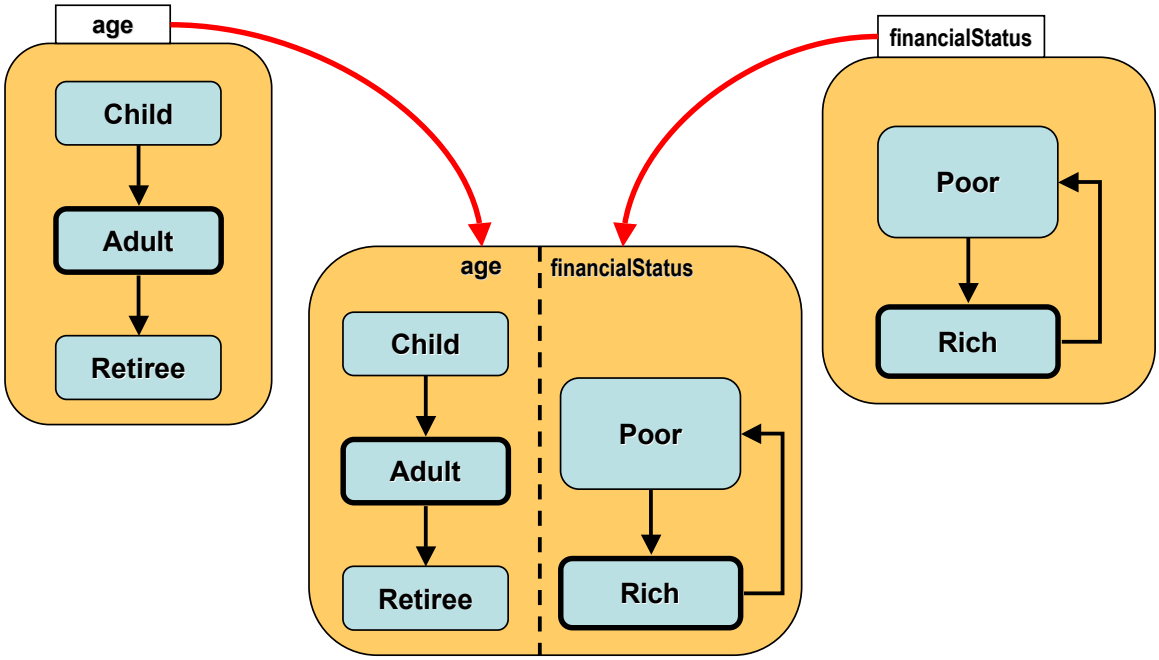
AND-States

- Gli stati possono essere decomposti in
  - OR-States
    - Uno stato (super-stato) può essere decomposto in un numero qialsiasi di OR-States
    - Quando l'oggetto si trova in un super-stato, deve trovarsi in UNO dei suoi OR-substates.

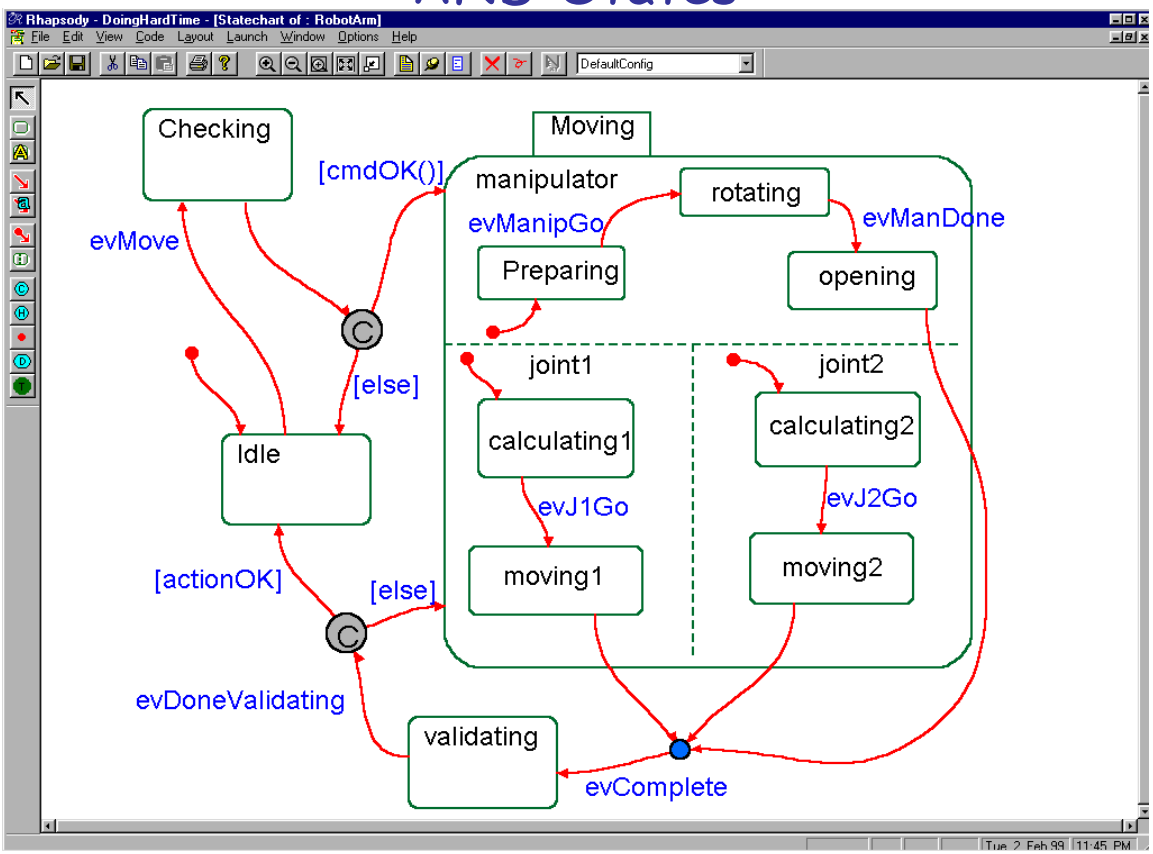
  oppure
  - AND-State
    - Uno stato (super-stato) può essere decomposto in un numero qialsiasi di AND-States (***regioni ortogonali***)
    - Quando l'oggetto si trova in un super-stato, deve trovarsi in TUTTI i suoi AND-substates attivi.
    - Regioni separate da linee tratteggiate

Regioni Ortogonali

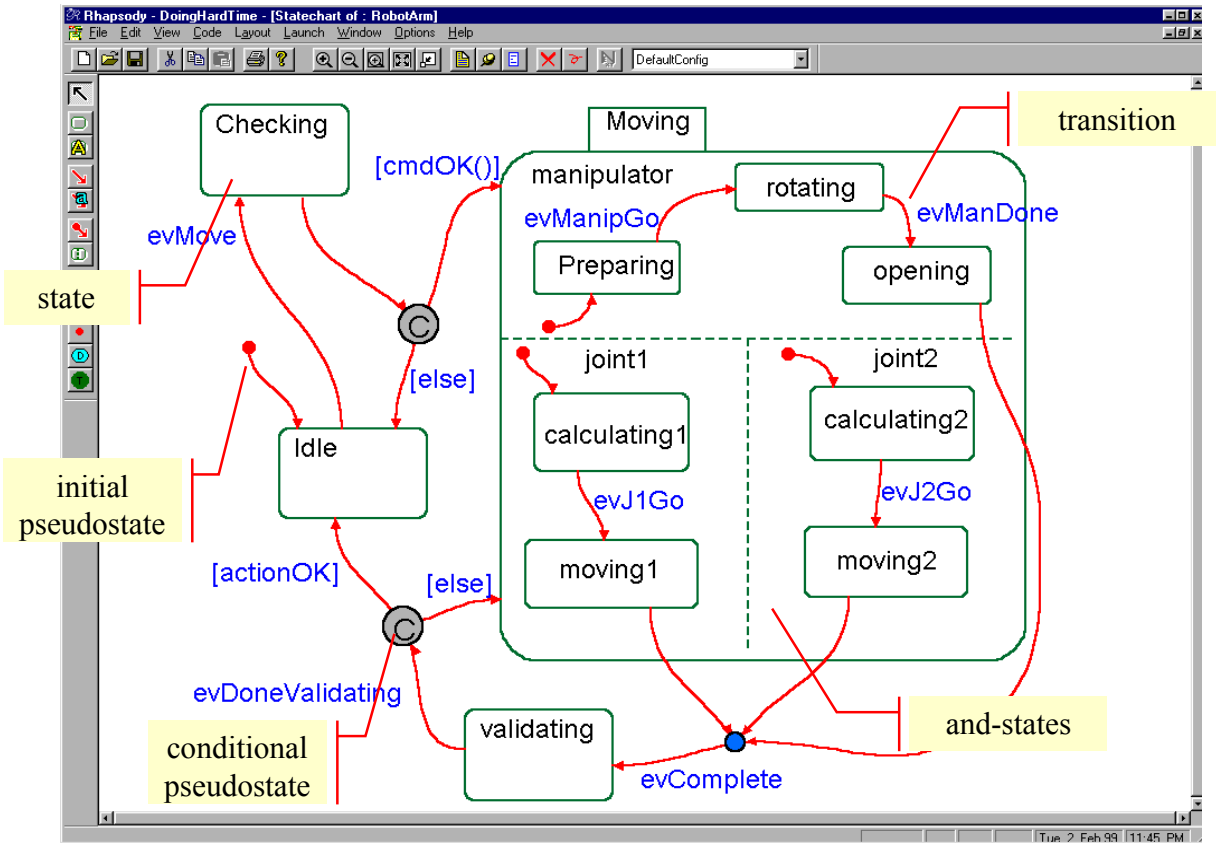
- Combinazione di descrizioni multiple simultanee



AND-States

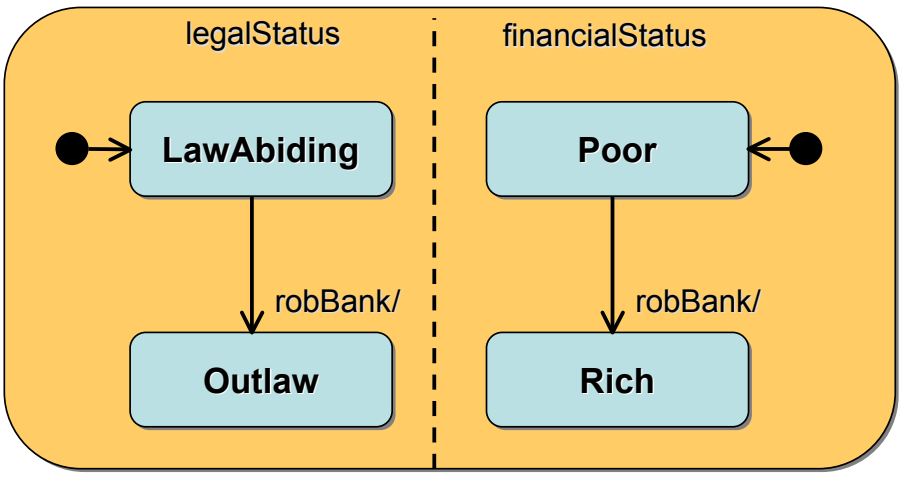


AND-States



Regioni Ortogonali - Semantica

- Tutte le regioni ortogonali reagiscono allo stesso evento, (in principo) simultaneamente.



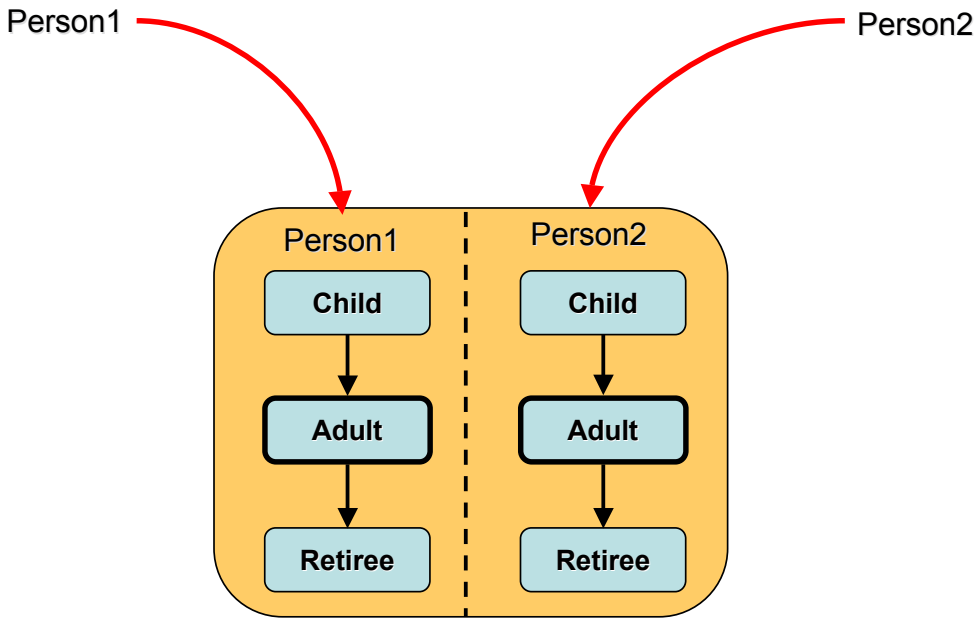
AND-States & Concorrenza

- Gli AND-states non sono necessariamente concorrenti
- Semantica degli AND-states tipicamente a ***interleaving***

- UML usa la concorrenza tra oggetti attivi come principale modo di modellare la concorrenza
- Gli AND-states possono essere implementati come threads concorrenti, ma questa non è l'unica strategia di implementazione corretta.

Uso concettualmente errato dell'ortogonalità

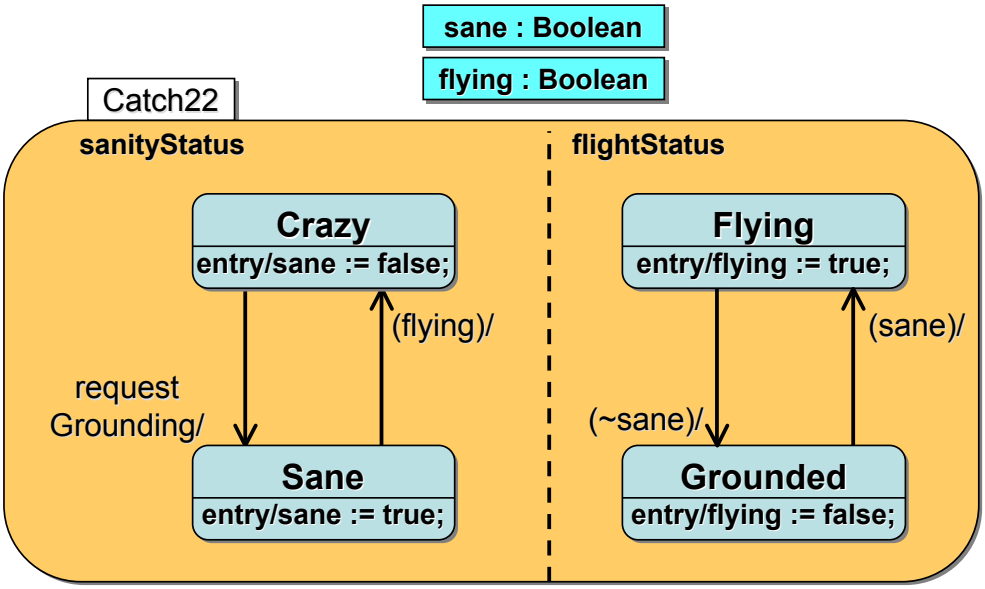
- Uso di regioni per modellare oggetti attivi independenti



- Soluzione: --> due istanze dello stesso oggetto attivo !!!

Interazioni tra Regioni

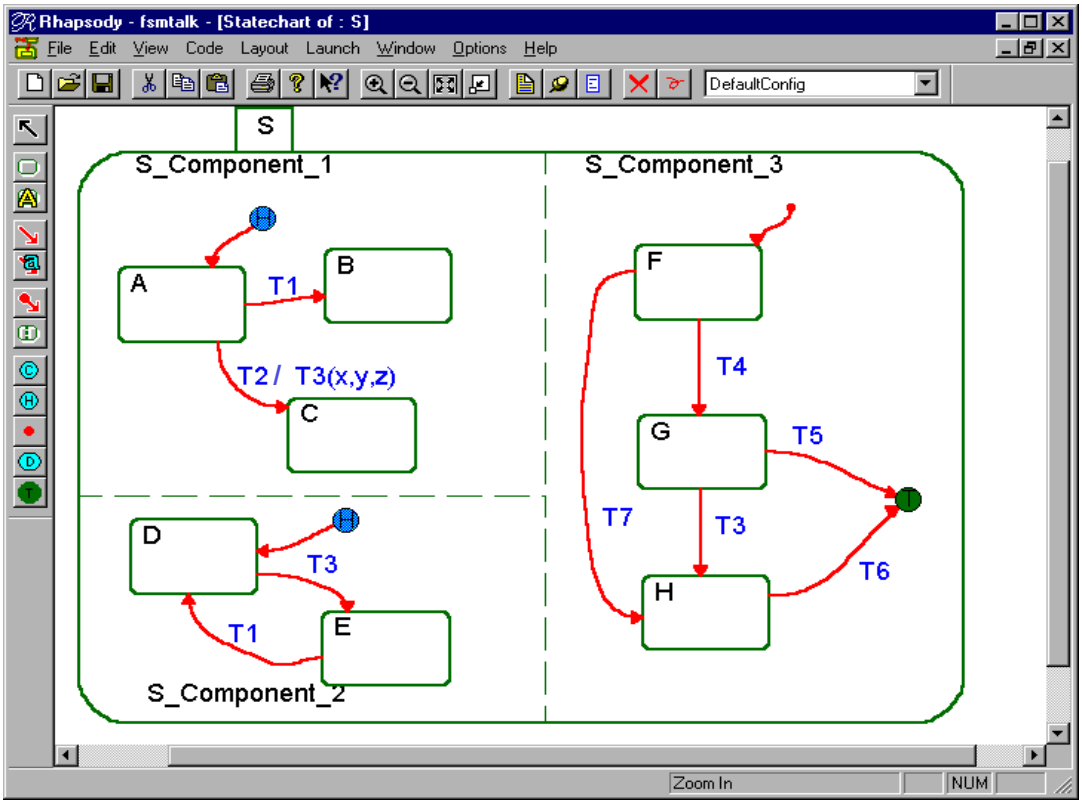
- Tipicamente, attraverso variabli condivise o riferimenti a cambio di stato delle altre regioni.



AND-State Communication

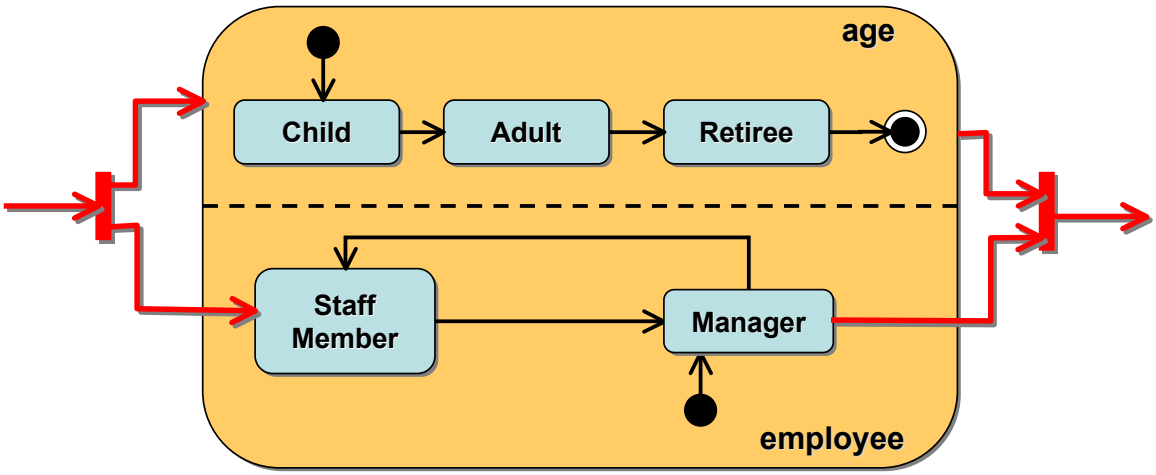
- Gli AND-states possono communicare via:
  - Broadcast events
    - Ricevuti da tutti gli AND-states attivi
  - Propagated events
    - Una transizione in un AND-state può inviare un evento sentito da un altro AND-state
  - Guardie
    - [IS_IN(state)] usa il substate di un AND-state in una guardia
  - Attributi
    - Siccome gli AND-states fano parte dello stesso oggetto, vedono tutti gli attributi dell'oggetto (variabili condivise)

Propagazione e Broadcast

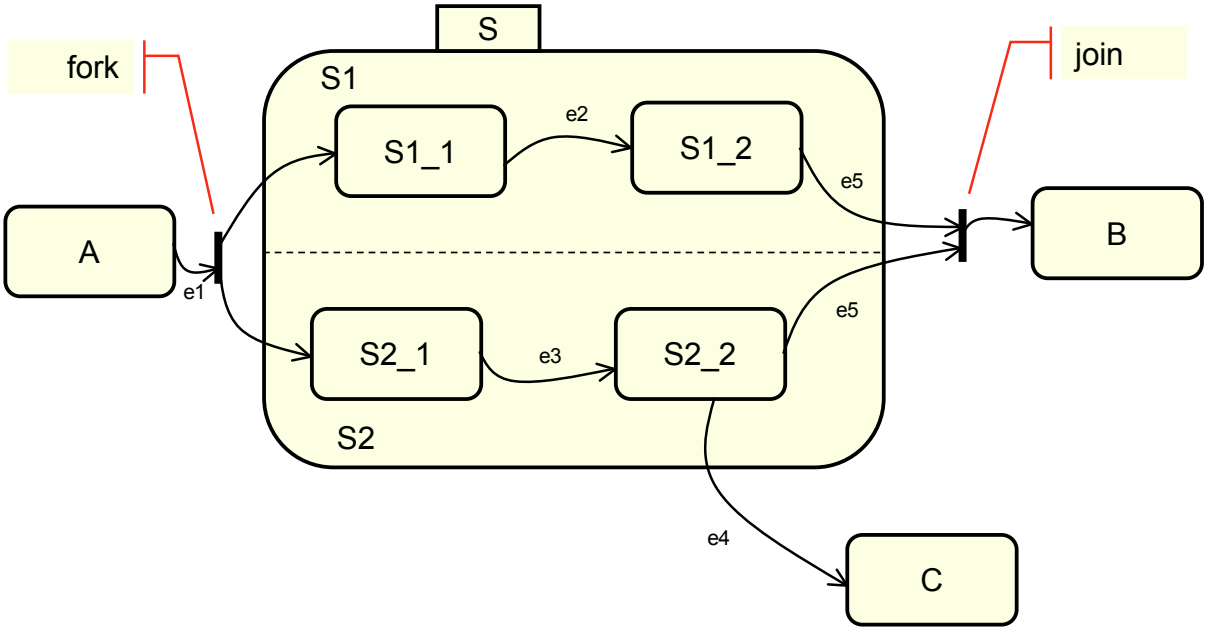


Fork e Join

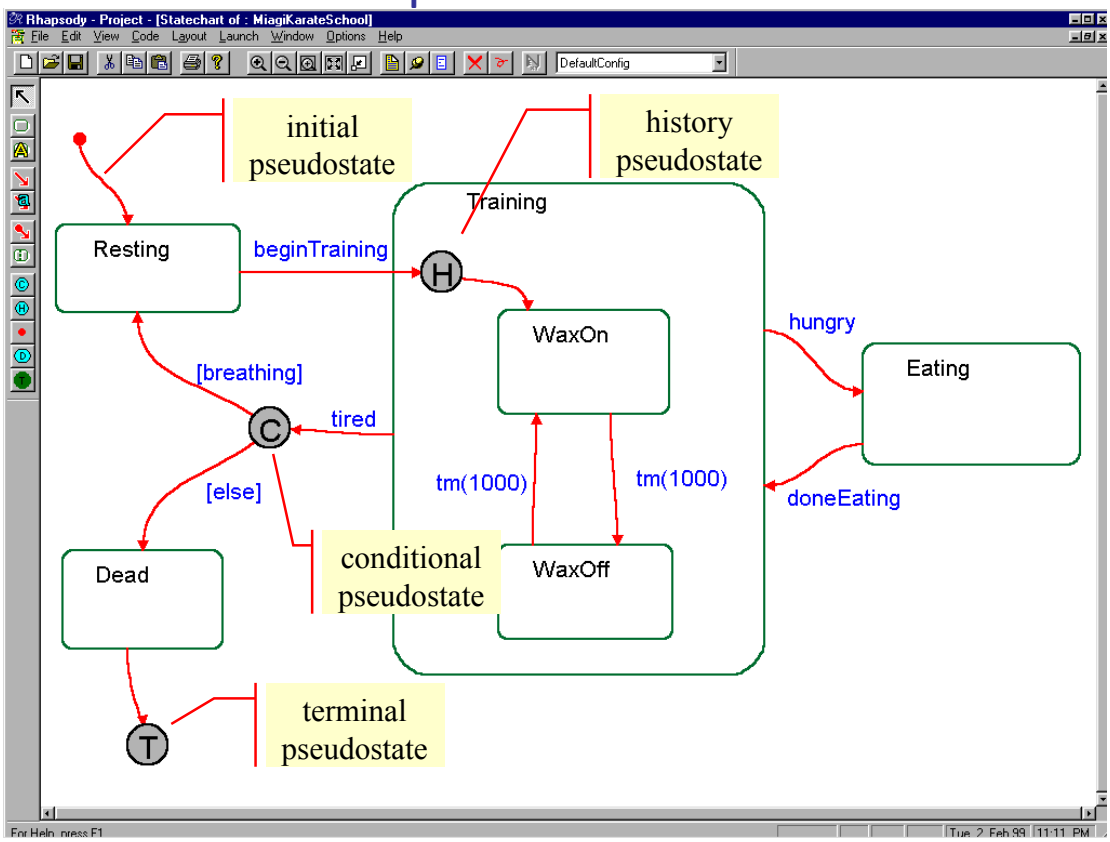
- Transizioni verso/da regions ortogonali:



Transizioni Complesse



pseudostates

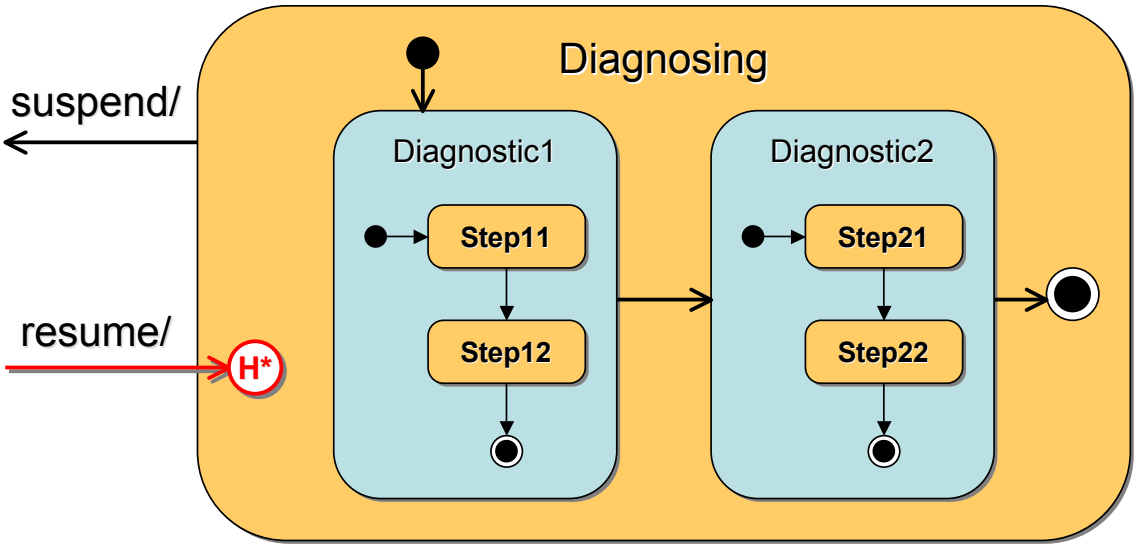


Pseudostates

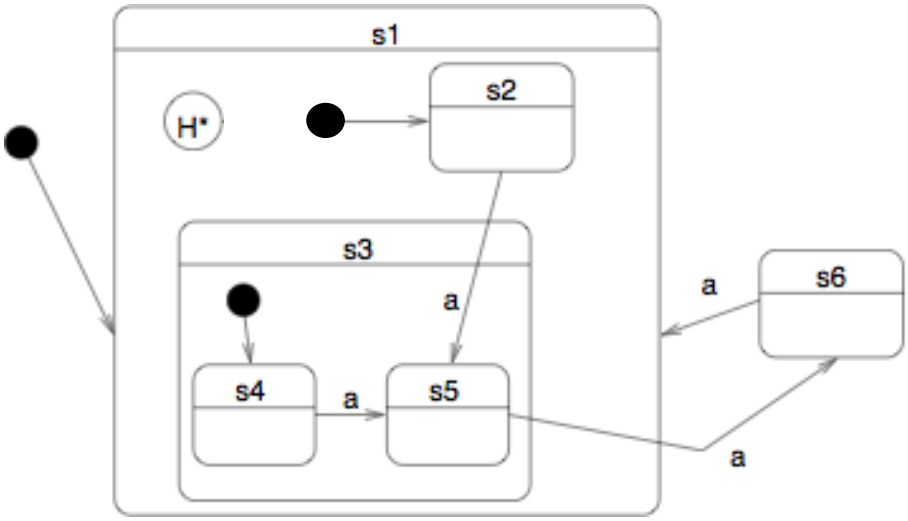
Symbol	Symbol Name	Symbol	Symbol Name
C or	Branch Pseudostate (type of junction pseudostate)	H	(Shallow) History Pseudostate
T or	Terminal or Final Pseudostate	H*	(Deep) History Pseudostate
* or n	Synch Pseudostate		Initial or Default Pseudostate
	Fork Pseudostate		Junction Pseudostate
	Join Pseudostate		Merge Junction Pseudostate (type of junction pseudostate)
[g] [g]	Choice Point Pseudostate	label	Stub Pseudostate

History

- Ritorno a uno stato gerarchico già visitato
  - deep /shallow history

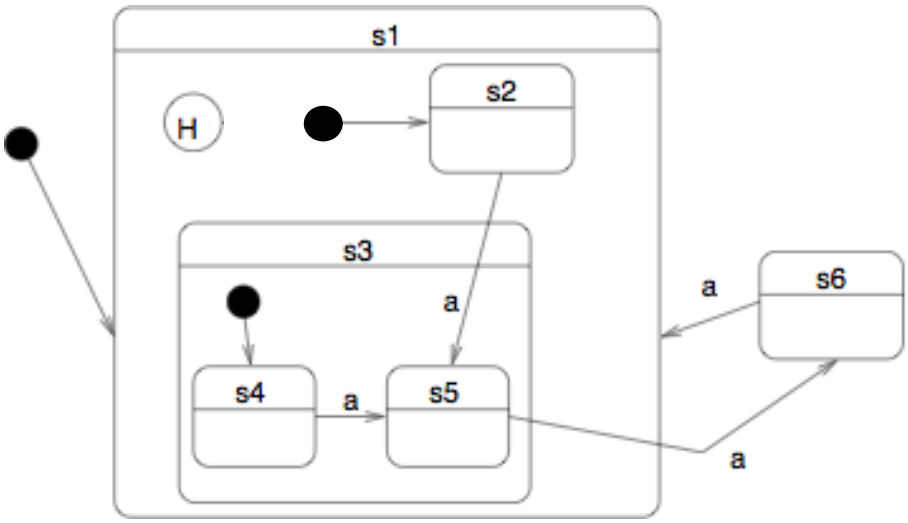


Deep History



deepHistory		
Step	Coda eventi	Stati attivi
1	[a, a, a]	s1, s2
2	[a, a]	s1, s3, s5
3	[a]	s6
4	[]	s1, s3, s5

Shallow History

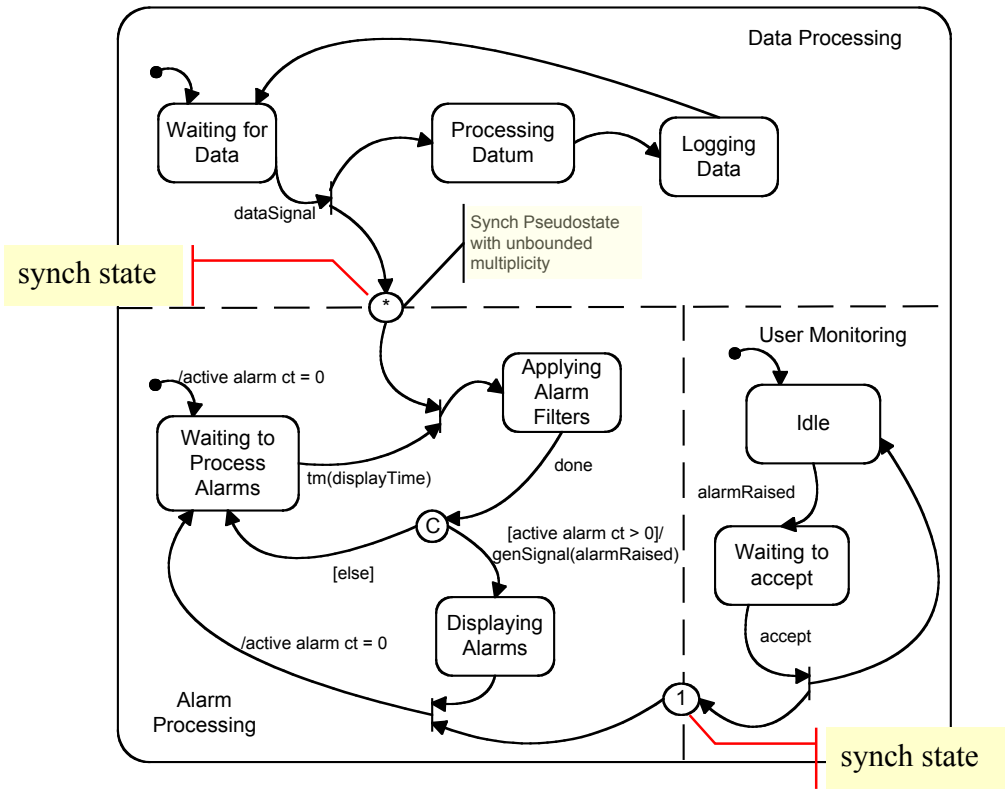


shallowHistory		
Step	Coda eventi	Stati attivi
1	[a, a, a]	s1, s2
2	[a, a]	s1, s3, s5
3	[a]	s6
4	[]	s1, s3, s4

Synch Pseudostate

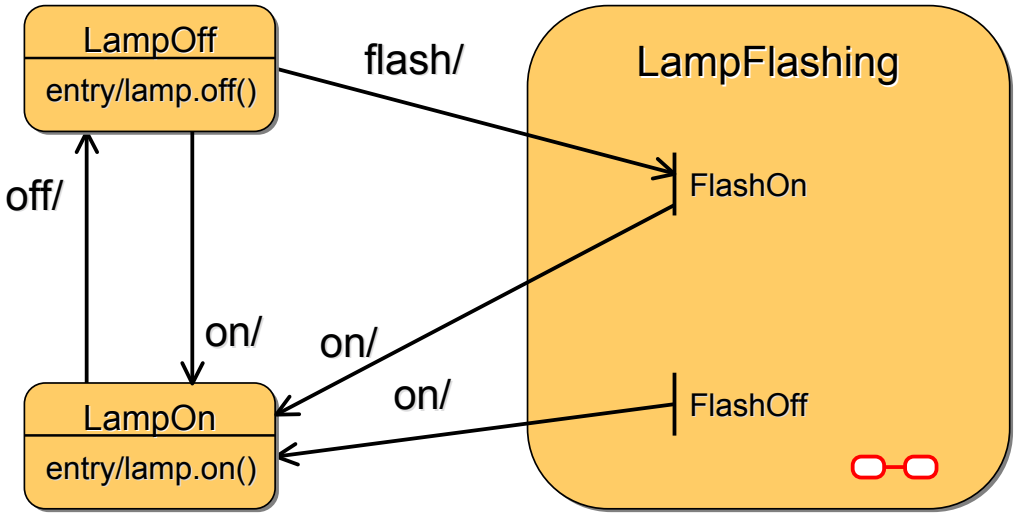
- Permette di far scattare una transizione solo quando un insieme di transizioni è avvenuto
- "Memorizza" che una o più specifiche transizioni sono avvenute,
- Simile al *posto* (piazza, place) di una Rete di Petri
- Permette di sincronizzare più AND-States

"Synch" Pseudostate

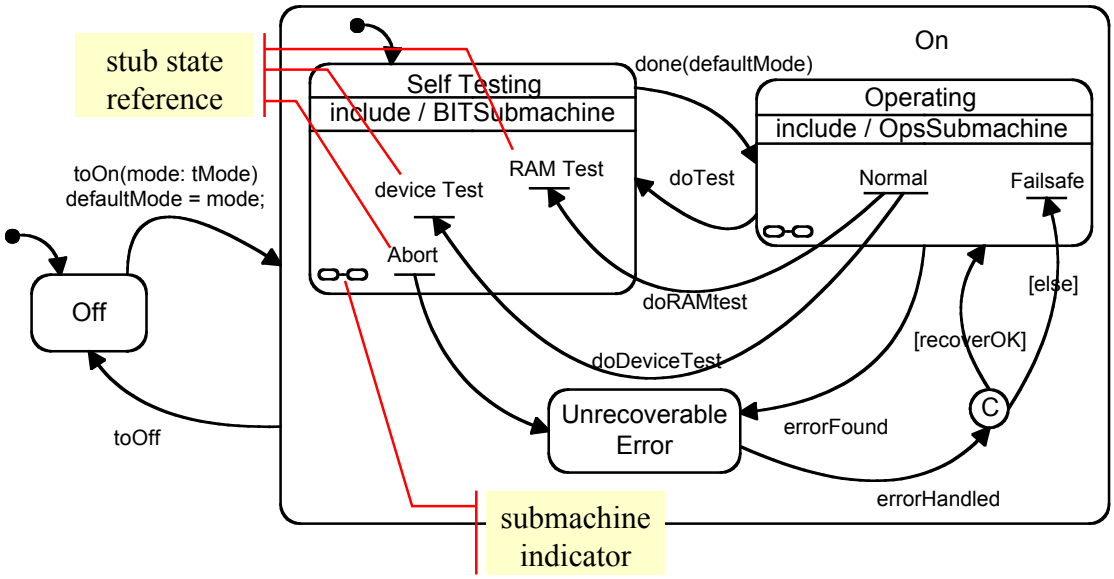


"Stub" Notation

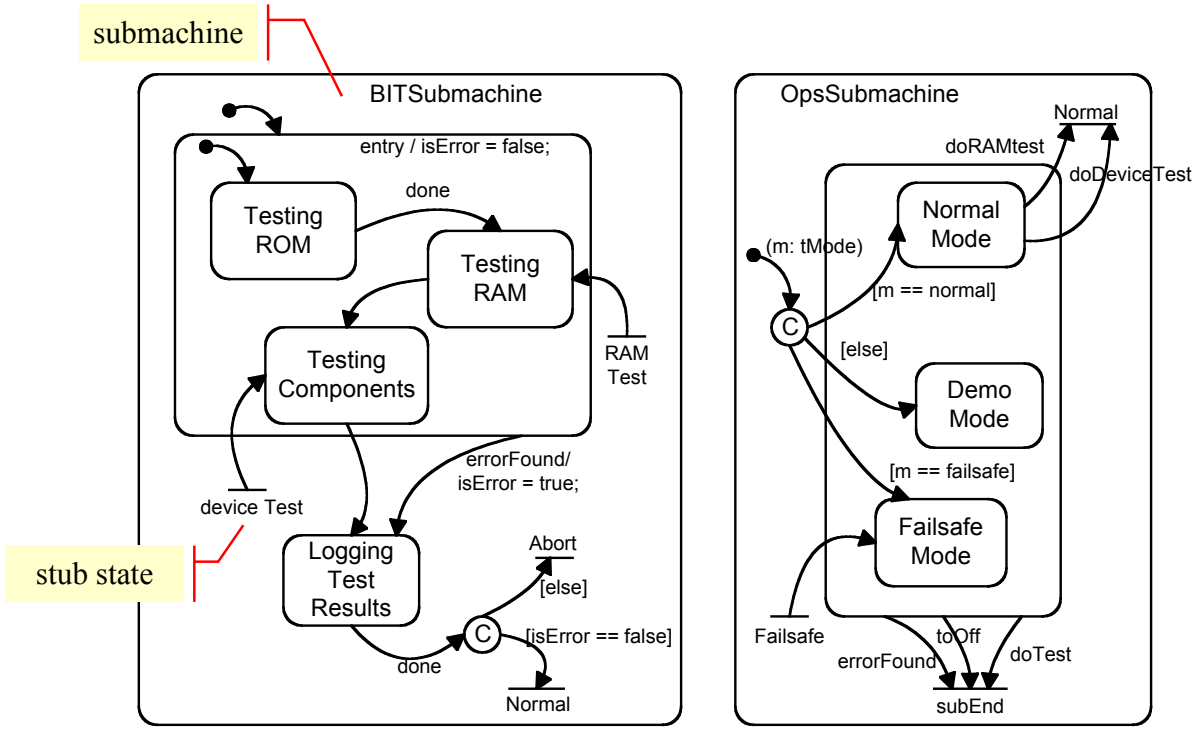
- Abbreviazione notazionale: nessuna aggiunta semantica



Submachines: Reference



Submachines: Referenced



Uso degli State Diagrams

- **Specifica** del comportamento di un sistema
  (necessita di uno strumento di editing)

- **Simulazione**
  (necessita di uno strumento di simulazione)
- **Verifica** (Model checking)
  (necessita di uno strumento di verifica)
- **Generazione automatica del codice**
  (necessita di un generatore di codice)

Credits

Bruce Powel Douglass, Real-Time UML, Slides, Ilogix

Gunnar Övergaard, Bran Selic, Conrad Bock, Behavioral Modeling, Object Modeling with OMG UML Tutorial Series, UML Revision Task Force, Nov. 2000