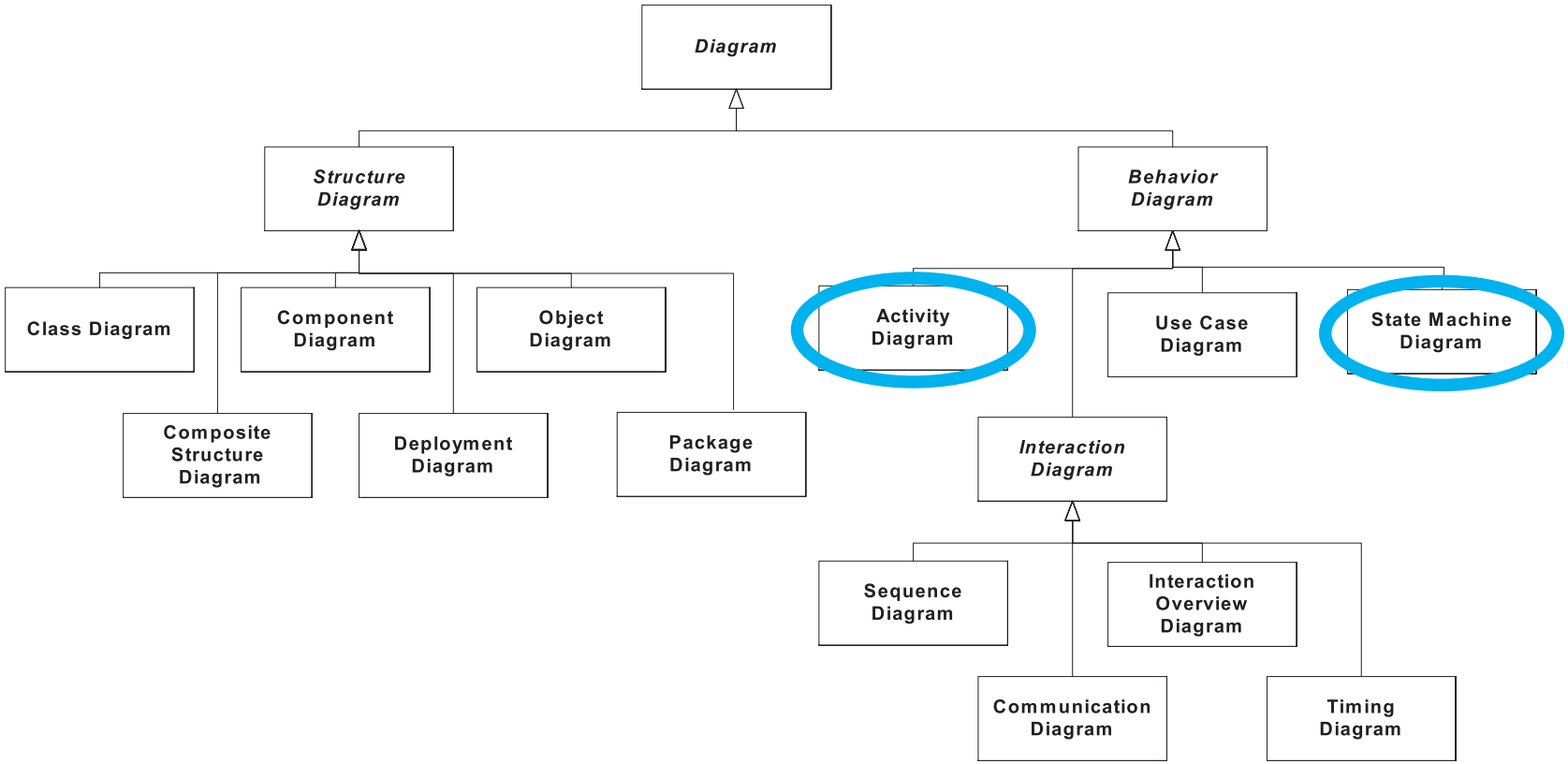
I diagrammi di attività e stato

Francesco Poggi
(dal materiale del prof. Ciancarini e dei dott. Di Iorio e Favini)

A.A. 2017-2018

Tassonomia dei diagrammi UML 2

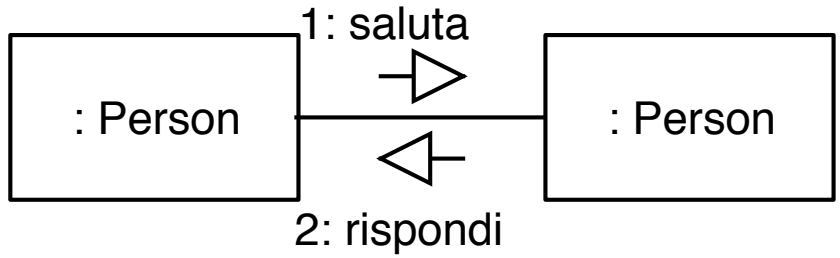


Cosa sono e a cosa servono

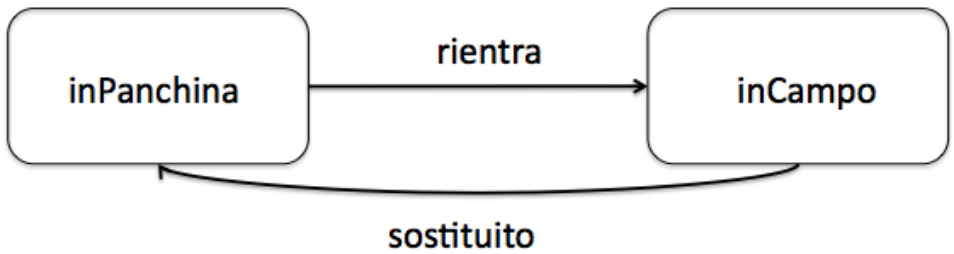
- I diagrammi di attività (*activity diagram*) e stato (*state machine diagram*) sono diagrammi che descrivono **comportamento**.
- Il diagramma di attività modella un comportamento (che riguarda **una o più entità**) come un **insieme di azioni** organizzate secondo un **flusso**.
- Il diagramma di stato modella il comportamento (generalmente di **una sola entità**) come **variazioni del suo stato interno**.

Un passo indietro: Interazione vs. Macchina a stati

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

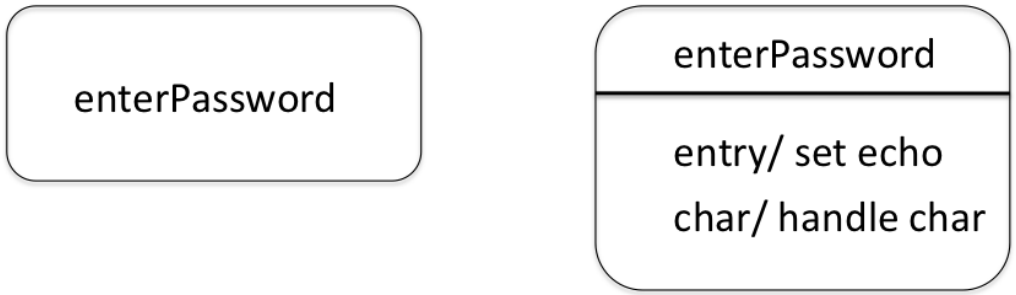


- **Macchina a stati**: descrive la sequenza di stati in cui si trova un oggetto durante il suo ciclo di vita e in risposta a eventi



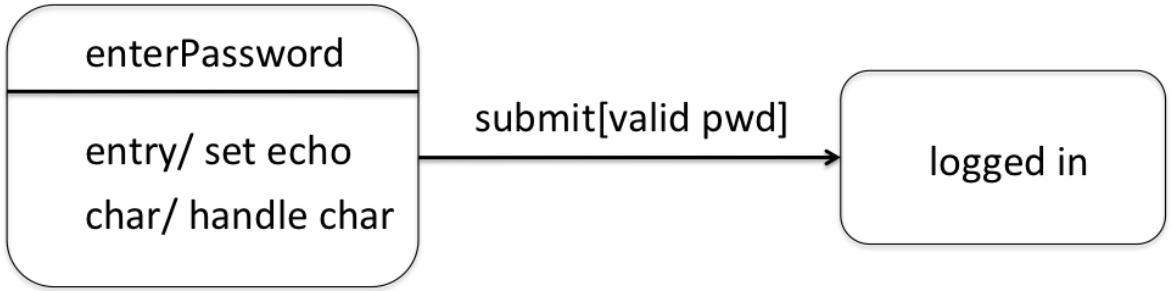
Macchine a stati in UML

- Qualunque classificatore UML può essere associato a una macchina a stati che descrive il funzionamento delle sue istanze.
- Uno **stato** è una condizione o situazione nella vita di un oggetto in cui esso:
  - soddisfa una condizione,
  - esegue un'attività o
  - aspetta un evento



Eventi e transizioni

- Un **evento** è 'la specifica di un'occorrenza che ha una collocazione nel tempo e nello spazio'.
- Una **transizione** è 'il passaggio da uno stato a un altro in risposta ad un evento'.



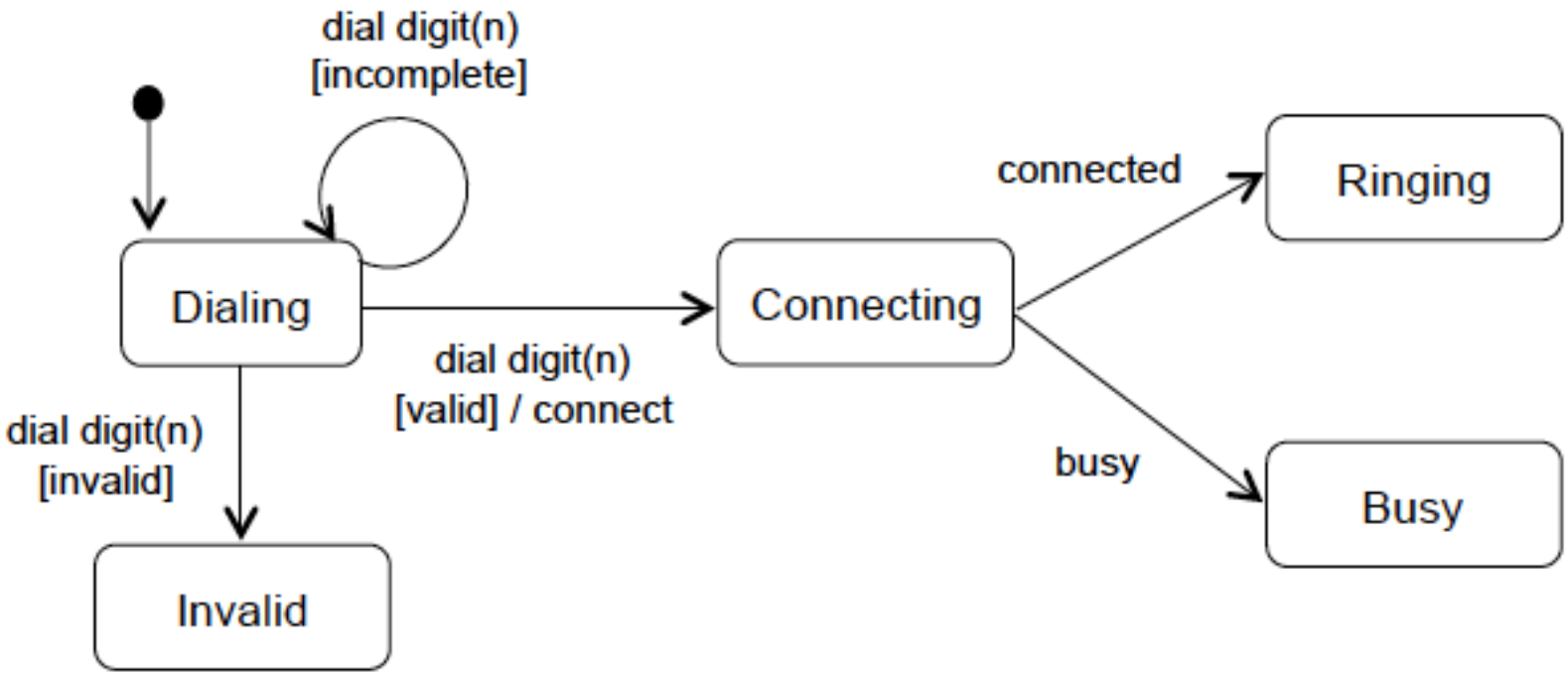
Semantica

- La macchina a stati riceve occorrenze di eventi che vengono salvati su una coda ed estratti uno alla volta.
- La semantica di questi eventi è di tipo **run-to-completion**: l'occorrenza di un evento viene estratta solo dopo che la macchina a stati ha finito di processare quella precedente.
- Se ci sono più transizioni eseguibili in un dato momento (es. 2 transizioni dallo stesso stato con lo stesso evento e due condizioni diverse, entrambe vere), solo una viene eseguita.

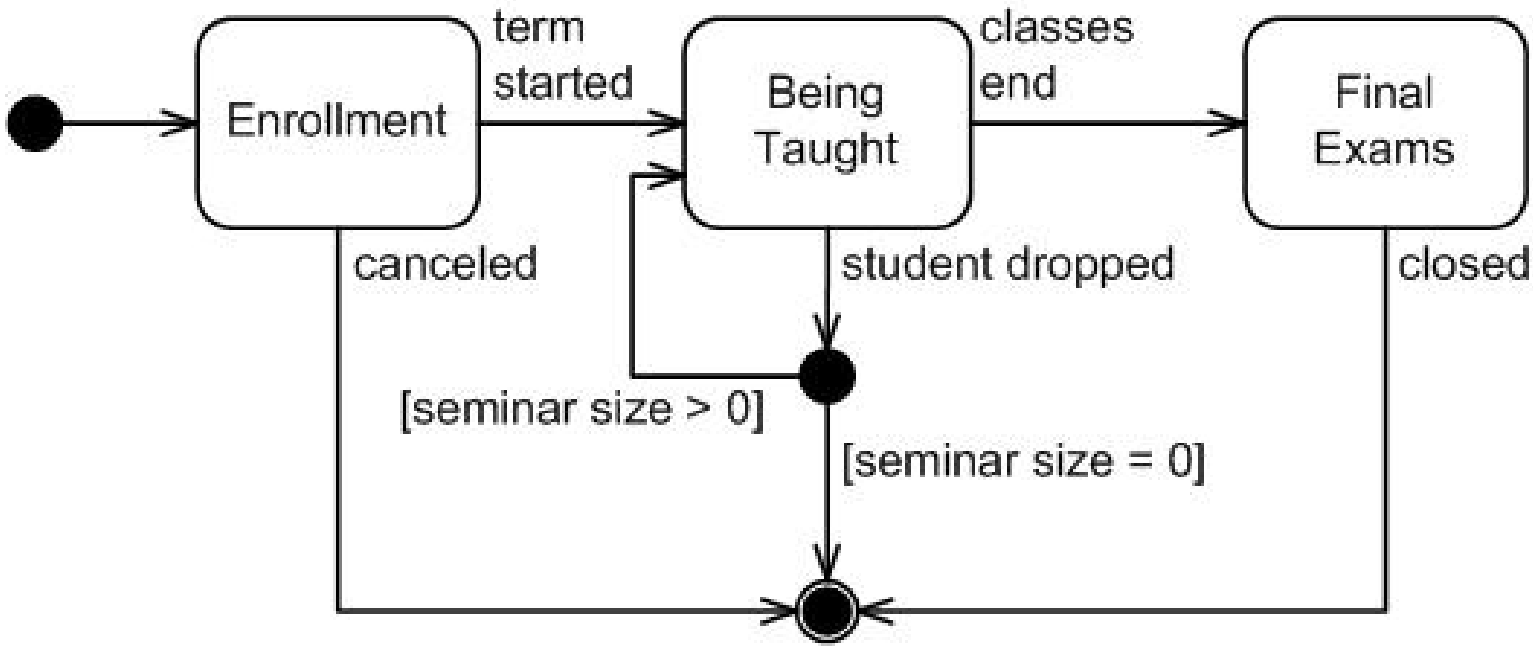
Transizioni

- Ogni transizione, oltre allo stato *origine* e *destinazione*, può specificare:
  - **Event**: un 'trigger' che attiva il passaggio di stato
  - **Guard**: una condizione che, se vera, permette il passaggio di stato
  - **Action**: un'azione che risulta dal combio di stato
- Sintassi: `event[guard]/action`
- La transizione avviene come risposta a uno degli eventi (quando la guardia è vera), e al momento della transizione il contesto esegue l'azione specificata
- Sono tutti **opzionali**

Un esempio completo (1)



Un esempio completo (2)

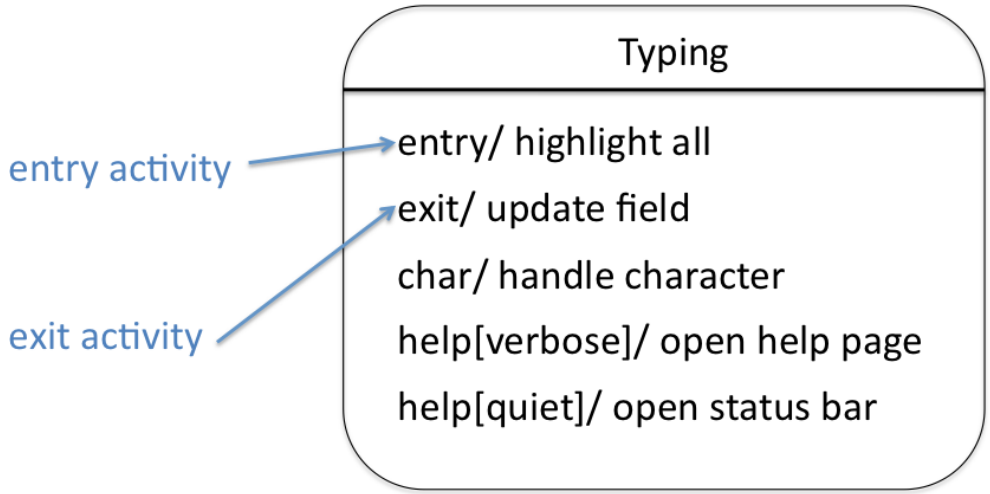


(pseudo)stati iniziali e finali

- Gli esempi precedenti mostrano due pseudo-stati, per avviare e bloccare la macchina a stati:
  - Il disco nero marca l'inizio dell'esecuzione. Non è uno stato vero e proprio ma un marcatore che punta allo stato da cui partire.
  - Il disco nero bordato (nodo finale), indica che l'esecuzione è terminata.
- Possono comparire in qualunque numero all'interno di un diagramma (o di uno stato composito, che vedremo in seguito).

[Azioni interne]

- Uno stato può reagire ad eventi (e verificare condizioni) anche senza una transizione ad uno stato diverso
- Le *internal activities* sono mostrate nel secondo slot e seguono la stessa sintassi delle transizioni
- Simile ad una *self-transition*
- Esempio: riempimento di un campo di testo in un form

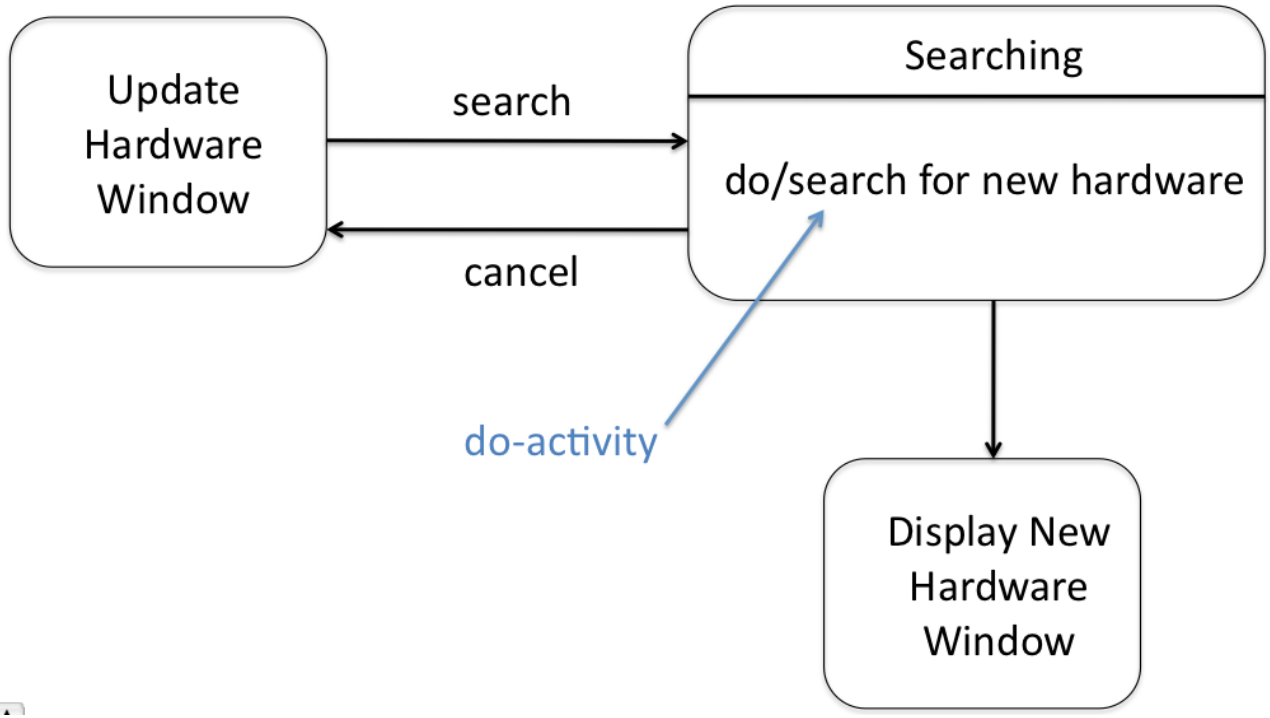


[Entry, Exit, Do]

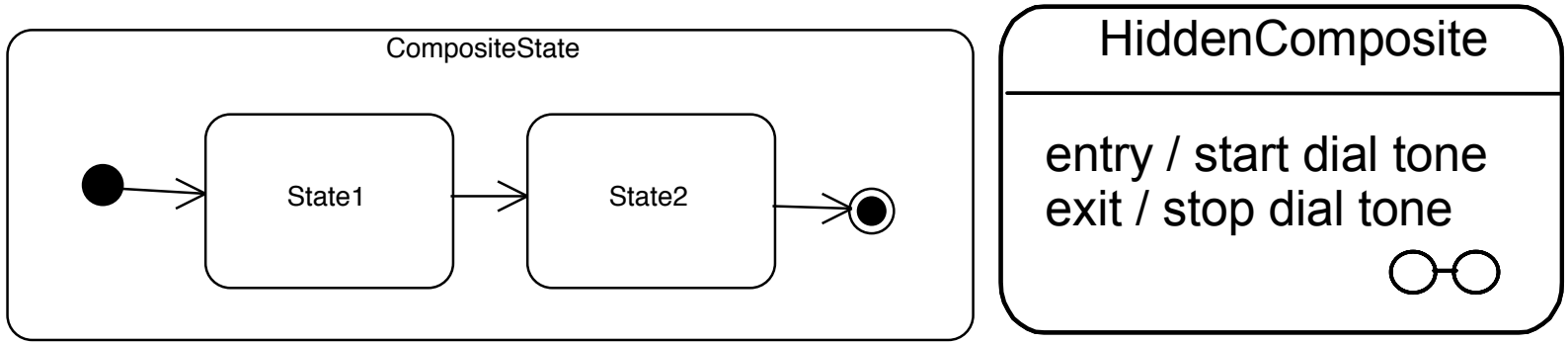
- All'interno di uno stato si possono usare alcune azioni speciali, indicate tramite keyword:
  - **entry**: eseguita quando l'oggetto entra nello stato
  - **exit**: eseguita quando l'oggetto esce dallo stato
  - **do** (*do-activity*): eseguita mentre l'oggetto è nello stato
- Una *self-transition* attiva sempre le *entry* ed *exit*, le *internal activities* invece no
- Una *do-activity* non è 'istantanea' ma può durare per un intervallo di tempo ed essere interrotta (da altri eventi)

[*do-activity*: esempio]

- Modelliamo la ricerca di nuovo hardware da installare su un sistema operativo.

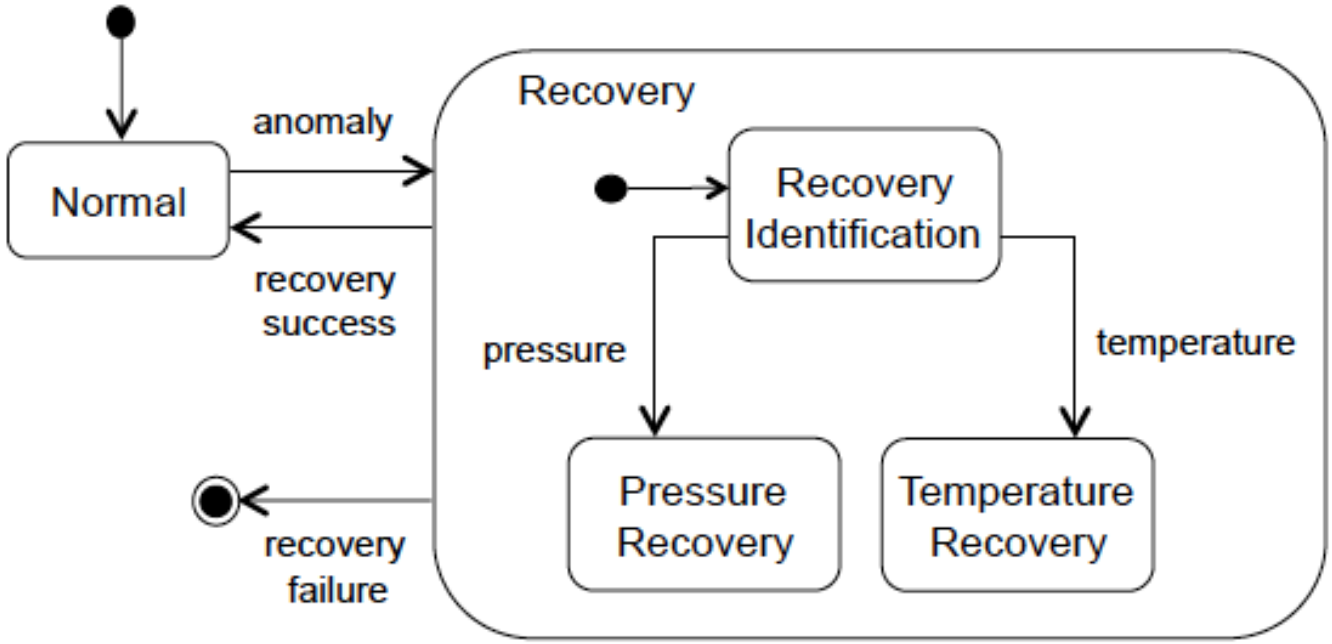


Stati compositi



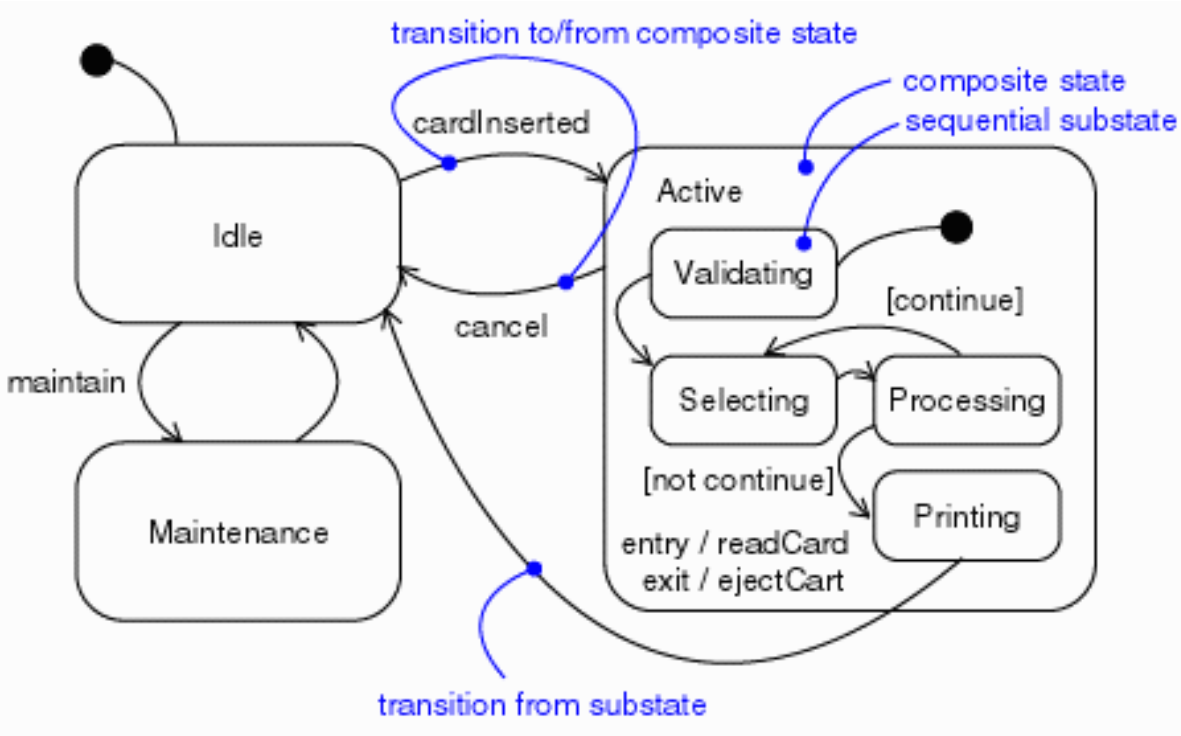
- Permettono di suddividere la complessità del modello: dall'esterno si vede un macro-stato, al cui interno vi sono altri stati.
- Si può anche creare uno stato che fa riferimento ad un'altro diagramma di macchina a stati (*submachine state*).
- Si può usare un'icona per rappresentare uno stato composito il cui comportamento interno non è mostrato.

Stati compositi: esempio



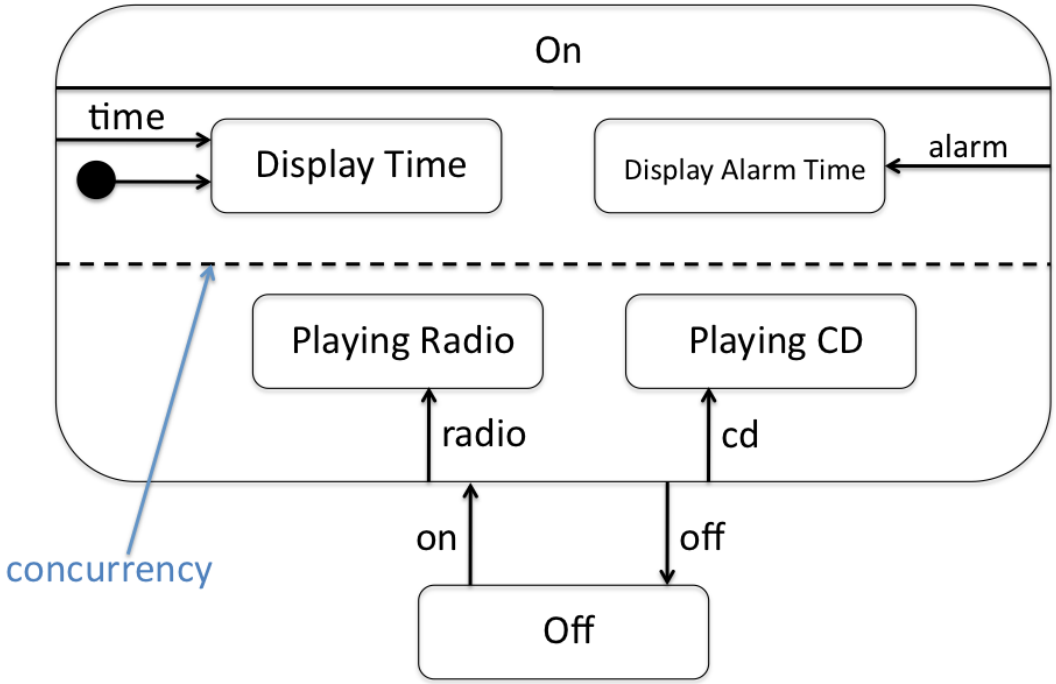
Stati compositi e transizioni

- Si possono definire transizioni da uno stato interno verso stati esterni
- Transizioni in uscita dal bordo dello stato composito e legate ad un evento sono ereditate da tutti gli stati all'interno.



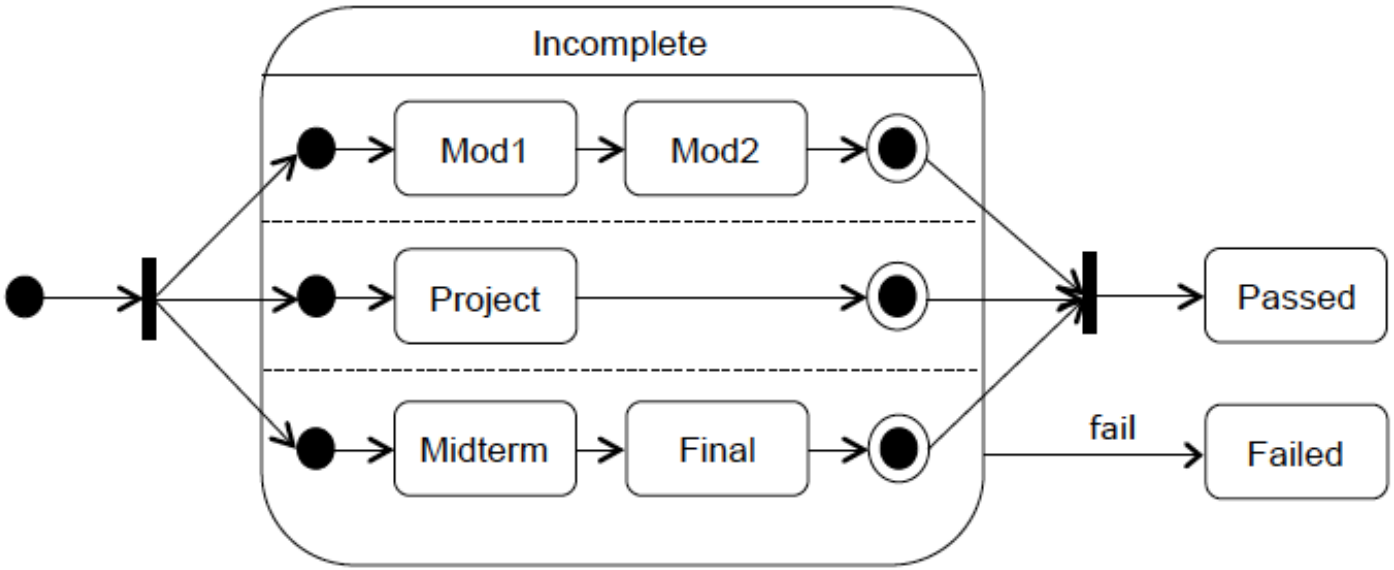
Stati compositi e concorrenza

- Gli stati compositi sono utili per modellare la concorrenza. Si divide lo stato composito in (sotto-)diagrammi **ortogonali** eseguiti in **mutua esclusione**
- Il diagramma sarebbe molto meno chiaro senza questo approccio (bisogna considerare tutte le possibilità)
- Esempio: una radiosveglia che o mostra l'orario o fa ascoltare musica



Stati compositi e sincronizzazione

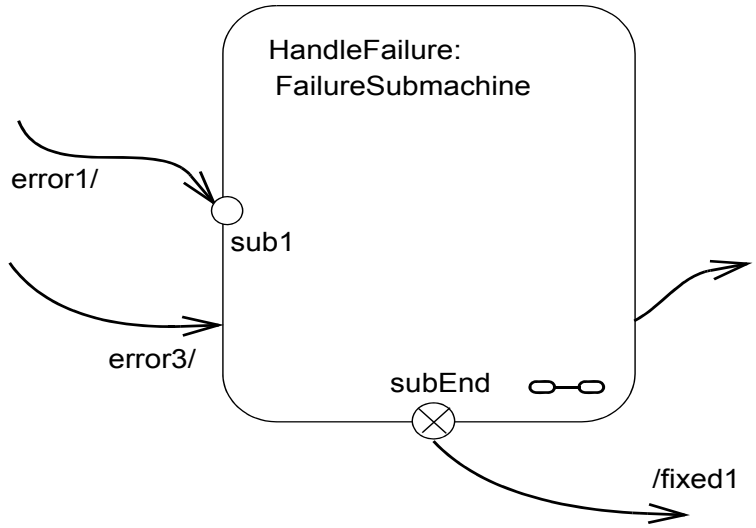
- Gli stati compositi sono inoltre utili per modellare la sincronizzazione. Si divide lo stato composito in (sotto-)diagrammi e si usano gli operatori di *fork* e *join* (che vedremo tra poco)
- Esempio: le attività e prove che uno studente deve superare per concludere un corso



Transizioni di completamento

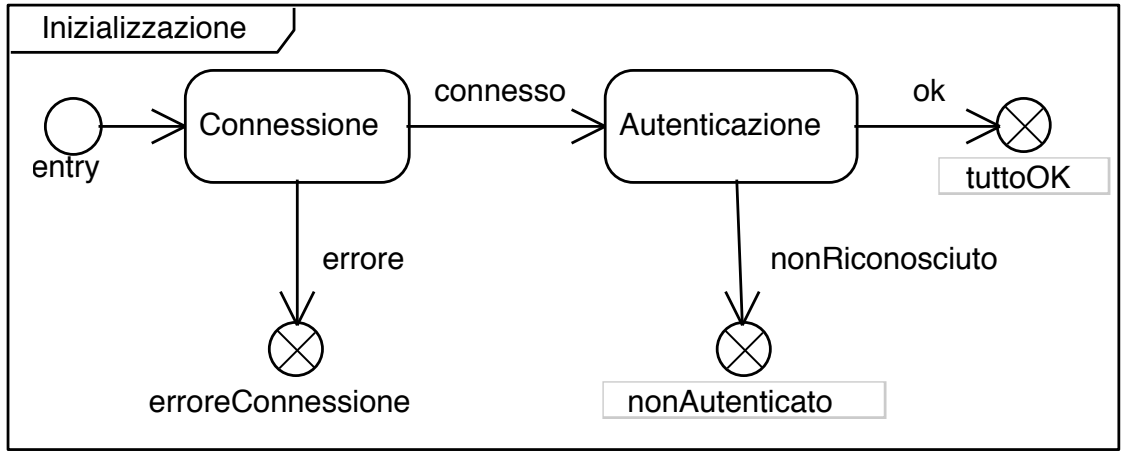
- Si tratta di transizioni che non hanno un evento associato (ma possono avere una guardia).
- Nel caso di uno stato semplice, una transizione di completamento è eseguita al termine dell'attività di quello stato (fine azioni *do*).
- Nel caso di uno stato composito o *submachine state* una transizione di completamento è eseguita quando si giunge in uno stato finale oppure un *exit point*.

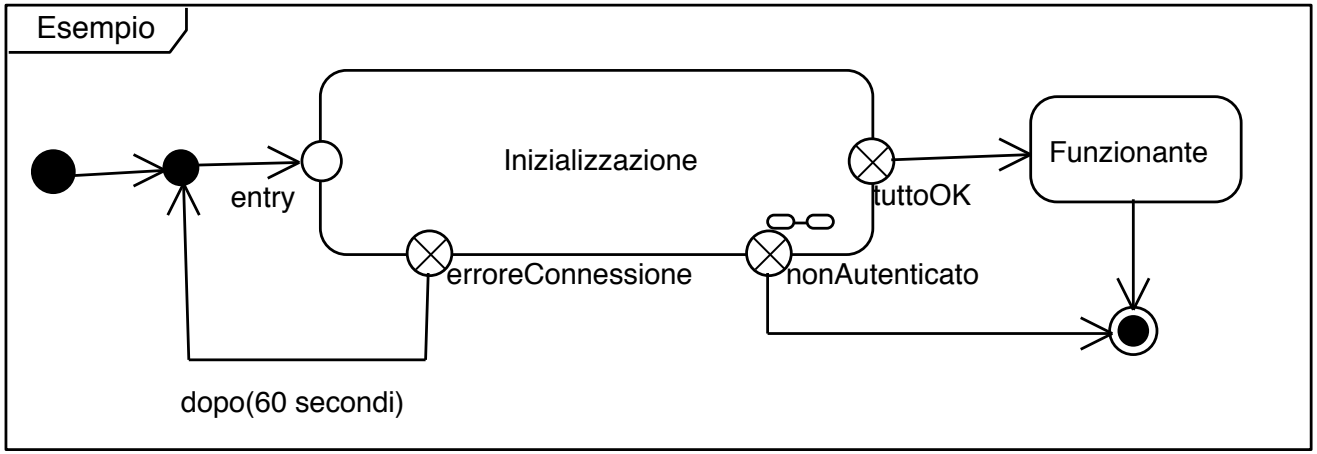
Completamento ed entry/exit points



- 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 di completamento che genera il comportamento *fixed1*.
- Se l'esecuzione termina nello stato finale si segue la transizione sulla destra.

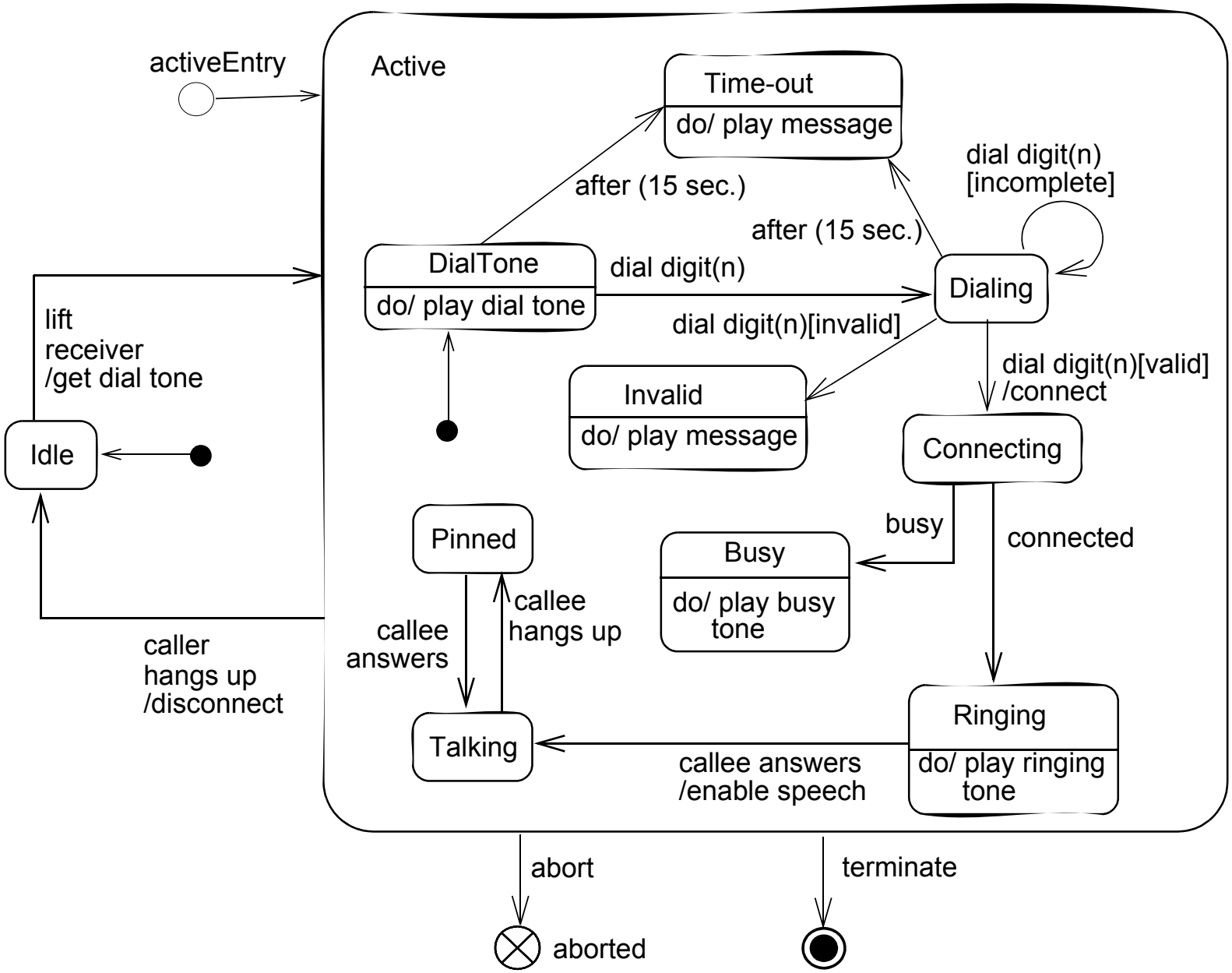
Entry/exit points (2)





- Forniscono un modo per entrare ed uscire dalle macchine a stati in diversi punti (simili a *entry/exit points* interni)
- Si possono usare in combinazione con i nodi iniziali e finali.

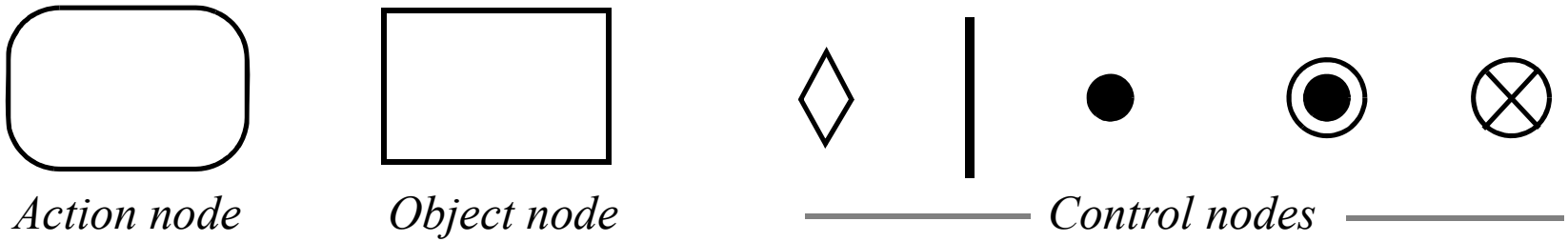
Un esempio completo



Il diagramma di attività

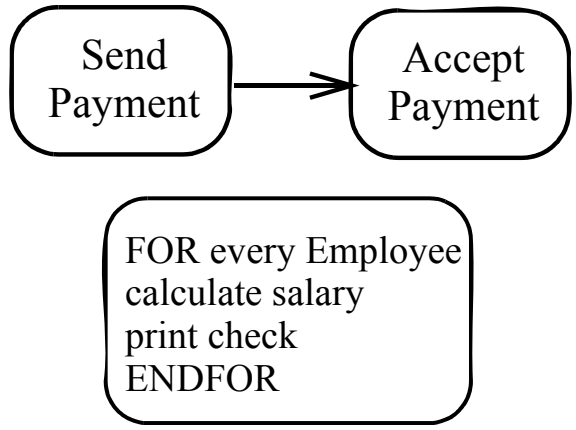
- Modella un'attività relativa ad un qualsiasi oggetto, ad esempio:
  - classi
  - casi d'uso
  - interfacce
  - componenti
  - interfacce
  - operazioni di classe
- Alcuni usi dei diagrammi di attività:
  - modellare il flusso di un caso d'uso (analisi)
  - modellare il funzionamento di un'operazione di classe (progettazione)
  - modellare un algoritmo (progettazione)

Attività: ingredienti



- Nodi **azione**: specificano unità di comportamento.
- Nodi **oggetto**: specificano oggetti usati come input e output di azioni.
- Nodi **controllo**: specificano il flusso dell'attività.

Nodi azione

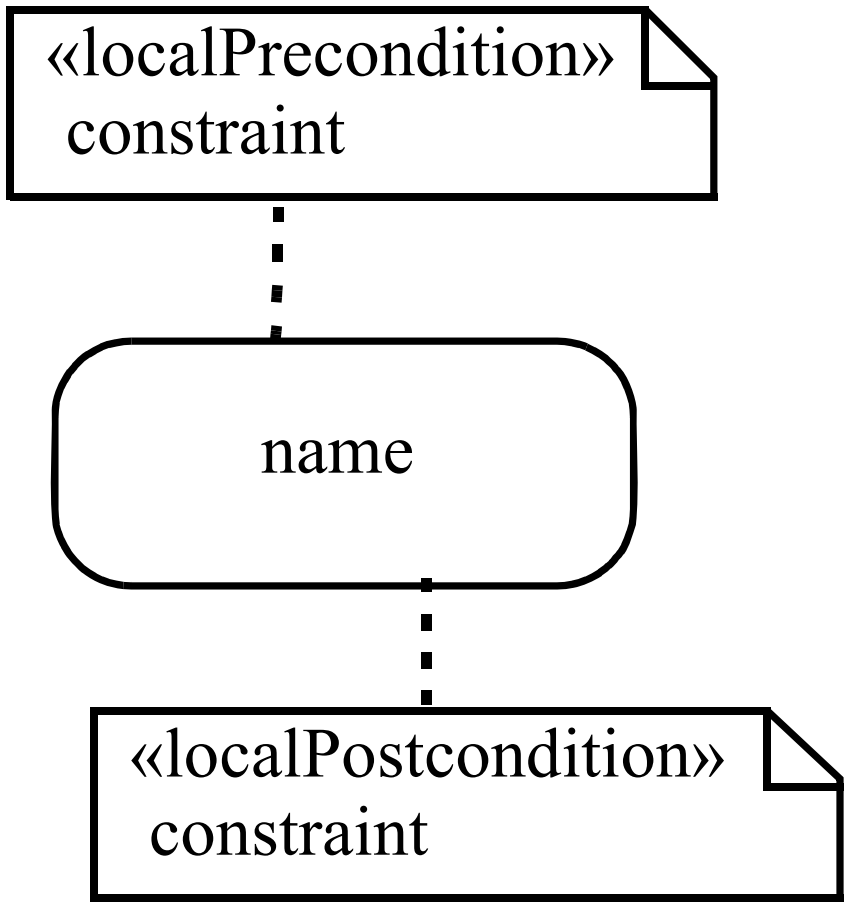


- Un'**azione** può invocare un'**attività**, un **comportamento** o un'**operazione**.
- Come gli altri elementi di UML, anche le azioni accettano livelli di dettaglio e linguaggi differenti.
- Al contrario dei messaggi nei diagrammi di interazione, le azioni non costringono il modellatore a definire tutte le entità in gioco.
- Le transizioni (frecce) tra azioni possono avere una guardia.

Transizioni e token

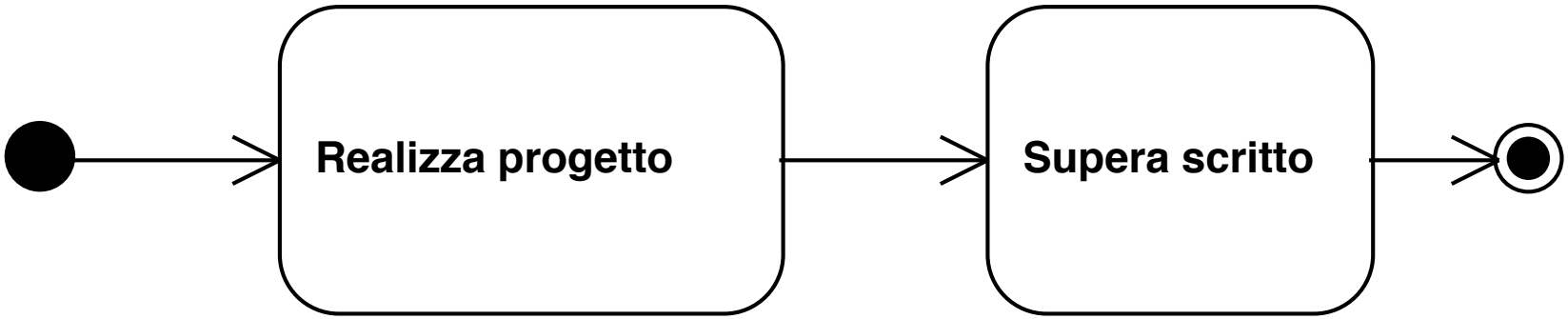
- Per capire la semantica dei diagrammi di attività, bisogna immaginare delle entità, dette **token**, che viaggiano lungo il diagramma.
- Il flusso dei token definisce il flusso dell'attività.
- I token possono rimanere fermi in un nodo azione/oggetto *in attesa* che si avveri una condizione su una freccia, oppure una precondizione o postcondizione su un nodo.
- Il movimento di un token è atomico.
- Un nodo azione viene eseguito quando sono presenti token su tutti gli archi in entrata, e tutte le precondizioni sono soddisfatte.
- Al termine di un'azione, sono generati *control token* su tutti gli archi in uscita.

Precondizioni e postcondizioni



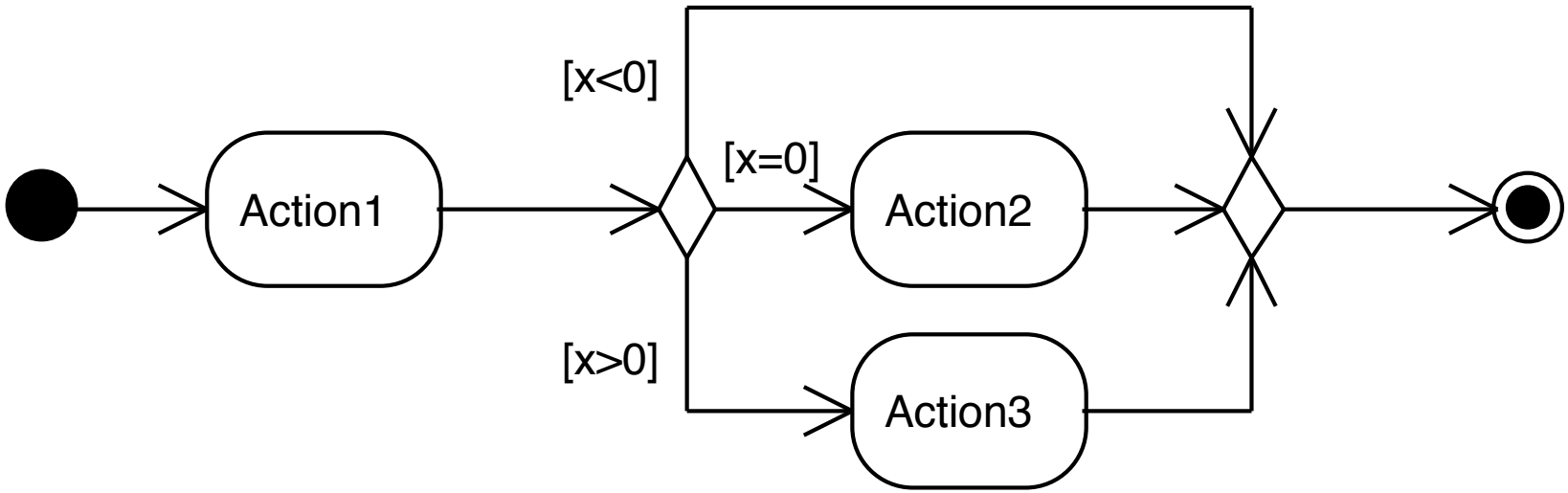
Si tratta di condizioni, espresse in qualunque modo, che devono essere soddisfatte per far iniziare o terminare l'azione (permettere a un token di entrare o uscire).

Nodi iniziali e finali



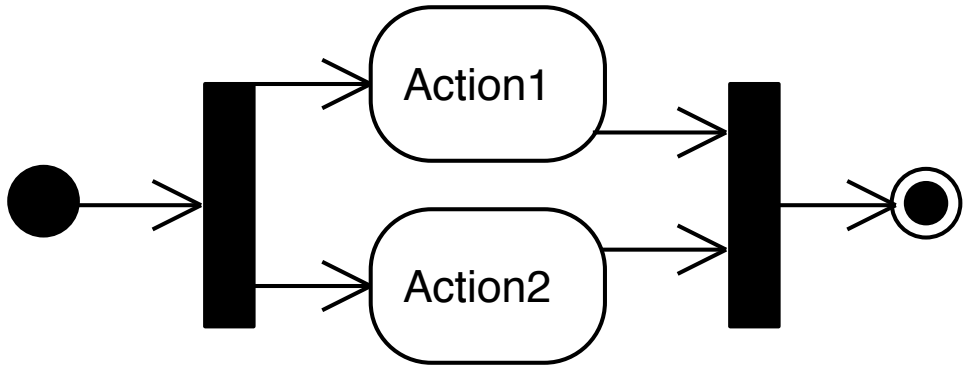
- Il disco nero marca l'inizio dell'attività (genera token).
- Quando un token raggiunge un disco nero bordato (nodo finale), l'attività ha termine.
- Possono comparire in qualunque numero all'interno di un'attività (ogni nodo iniziale fa partire un flusso di esecuzione, il primo nodo finale raggiunto ferma tutti i flussi).

Nodi decisione e fusione



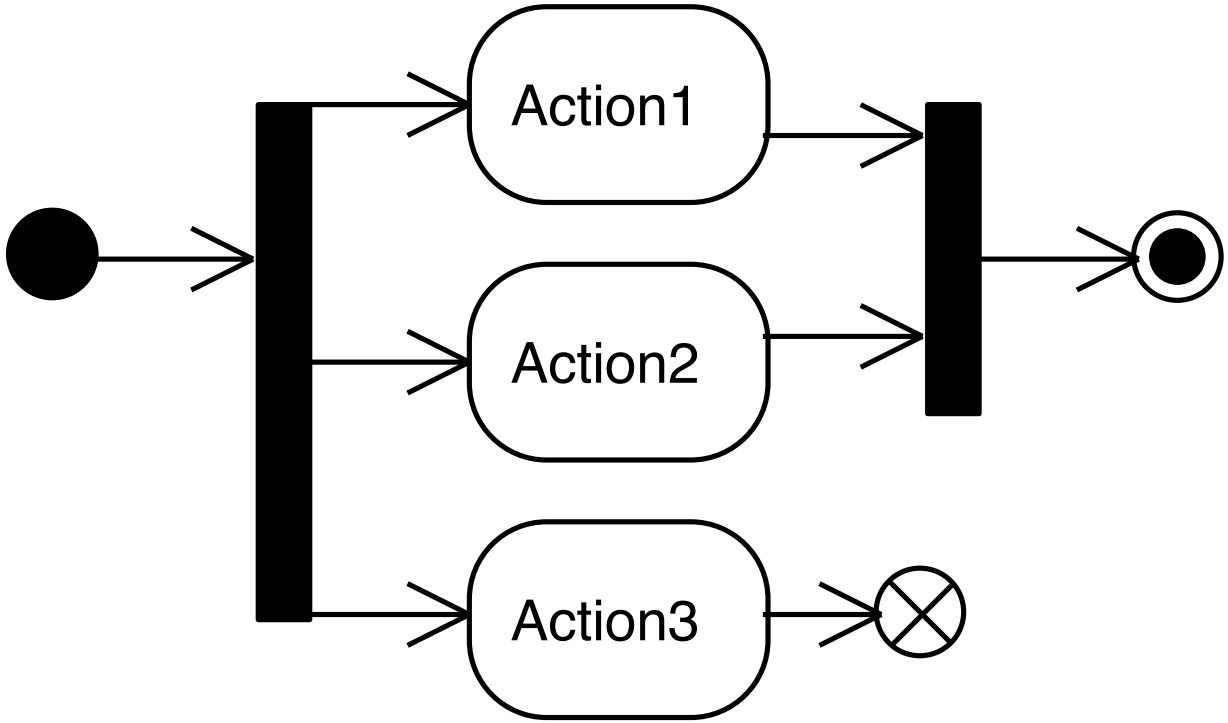
- I nodi **decisione** hanno un input e vari output **mutuamente esclusivi**: copiano i token in entrata su uno degli output.
- I nodi **fusione** hanno vari input e un solo output, sul quale vengono indirizzati tutti i token in ingresso.

Nodi fork/join



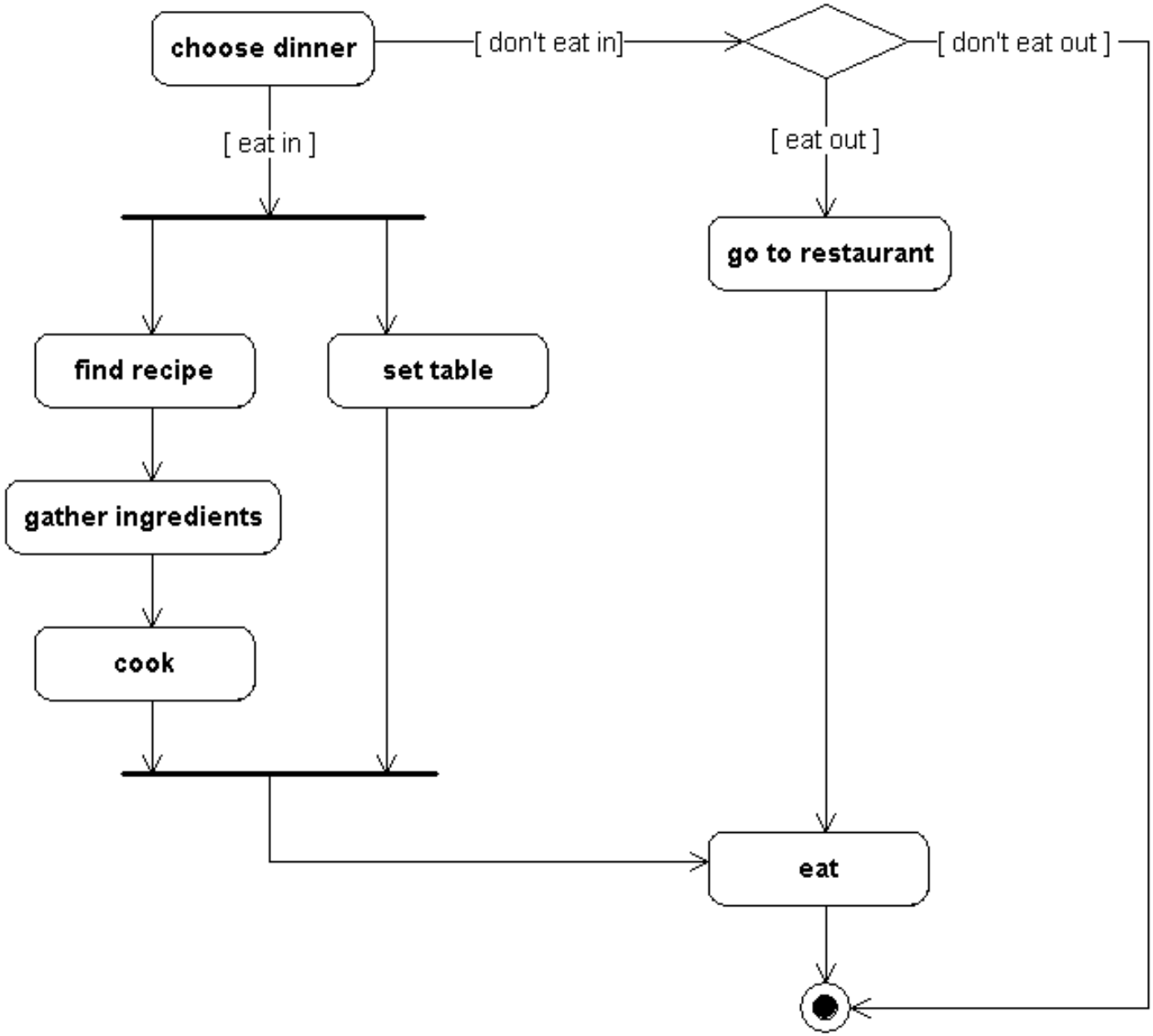
- I nodi **fork** hanno un ingresso e varie uscite: i token in ingresso sono duplicati su tutte le uscite.
- I nodi **join** hanno vari ingressi e una sola uscita: quando sono presenti token su tutti gli ingressi, viene prodotto almeno un token in uscita.
- I nodi fork dividono un'esecuzione in più flussi concorrenti, i nodi join sincronizzano e riuniscono i flussi.

Nodi finali di flusso

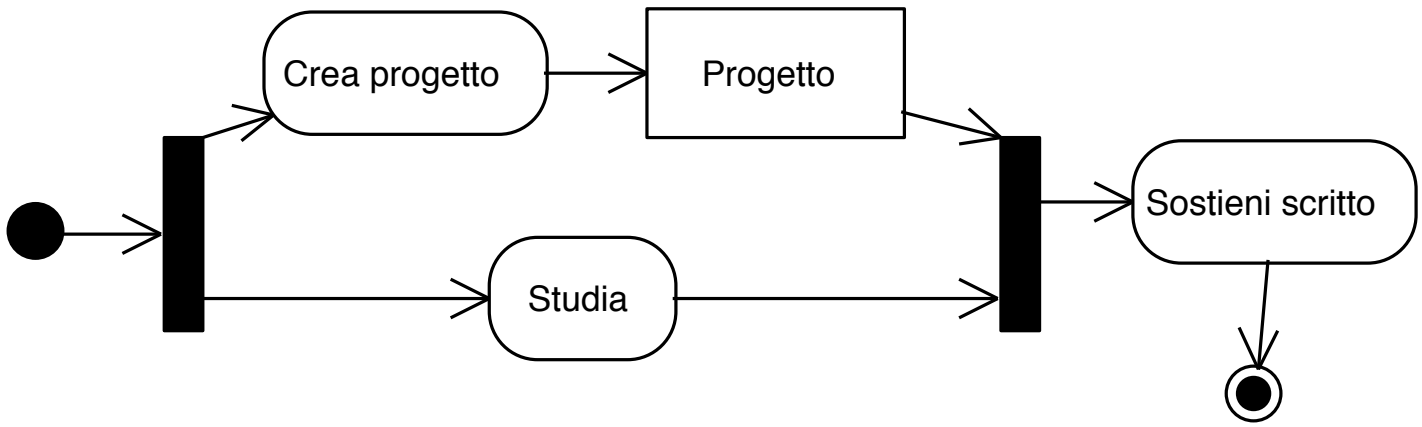


- Quando raggiunti da un token, causano la terminazione solo del flusso che li ha toccati.
- Il raggiungimento di un nodo finale di attività causa comunque la terminazione di tutti i flussi.

Un esempio completo



Nodi oggetto

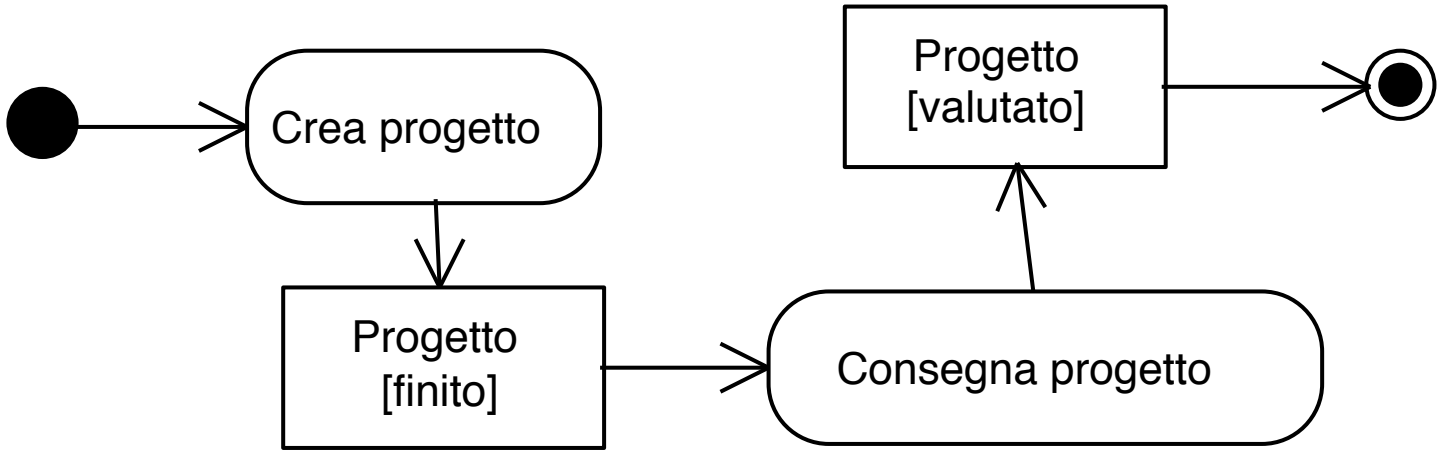


- Servono per modellare gli oggetti in input e output delle azioni
- I token in uscita da questi nodi sono *object token*, e sono diversi dai *control token* prodotti dai nodi azione: rappresentano veri e propri oggetti.
- Gli archi in entrata e uscita dai nodi oggetto sono *object flow* anziché *control flow*, e ci sono regole che limitano il loro uso (es: gli archi che entrano ed escono dai nodi decisione e fusione devono essere o tutti object o tutti control)

[Pin]

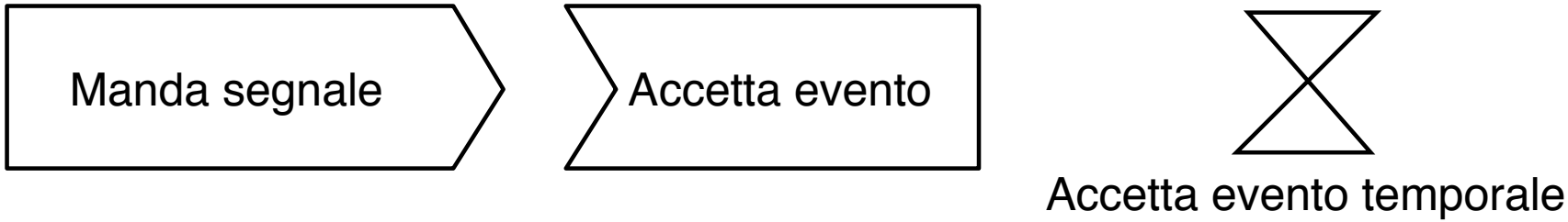
- Si agganciano ai nodi azione per definire un input oppure un output di quell'azione.
- Questa notazione è equivalente a quella di un nodo oggetto tra i due nodi azione.
- I pin aiutano a mostrare i parametri e valori di ritorno di un'azione.

Stato degli oggetti



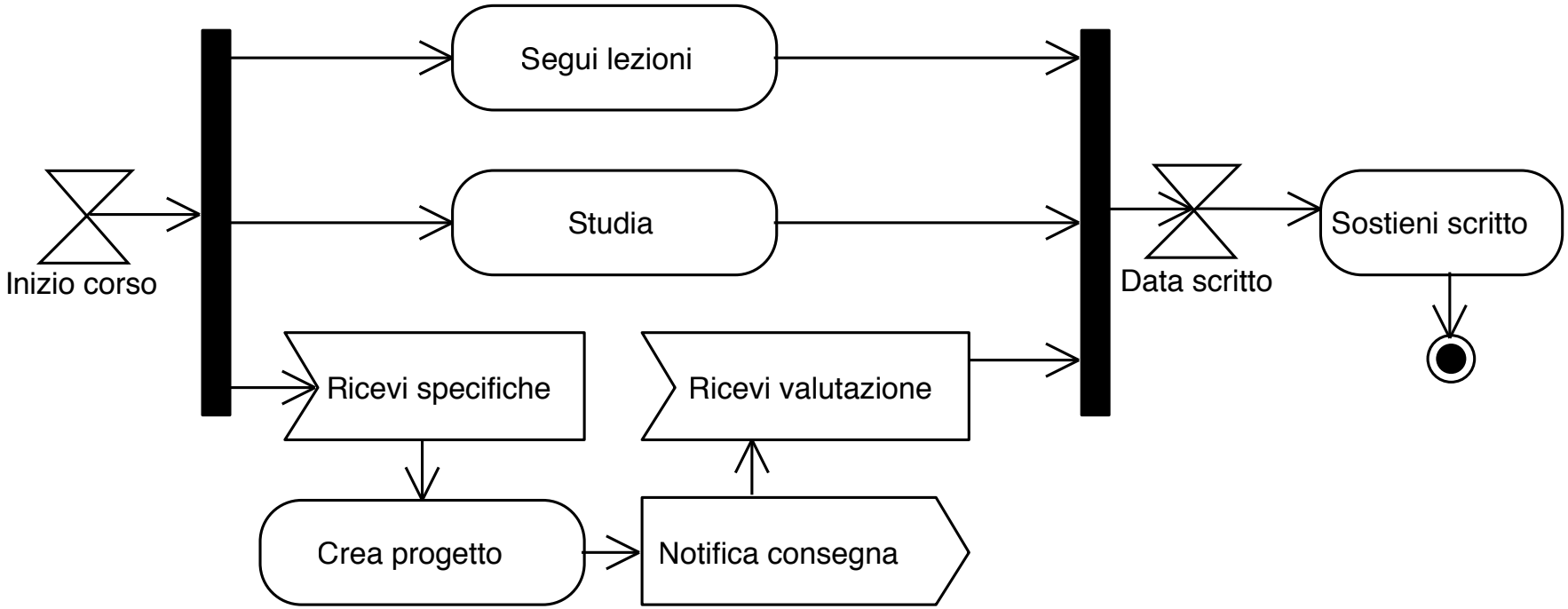
- Spesso risulta conveniente aggiungere lo stato di un oggetto per mostrarne l'evoluzione durante l'attività.
- Gli stati devono essere coerenti con la macchina a stati associata all'oggetto.
- Questo è l'anello di congiunzione tra diagrammi di attività e stato.

Segnali ed eventi (1)



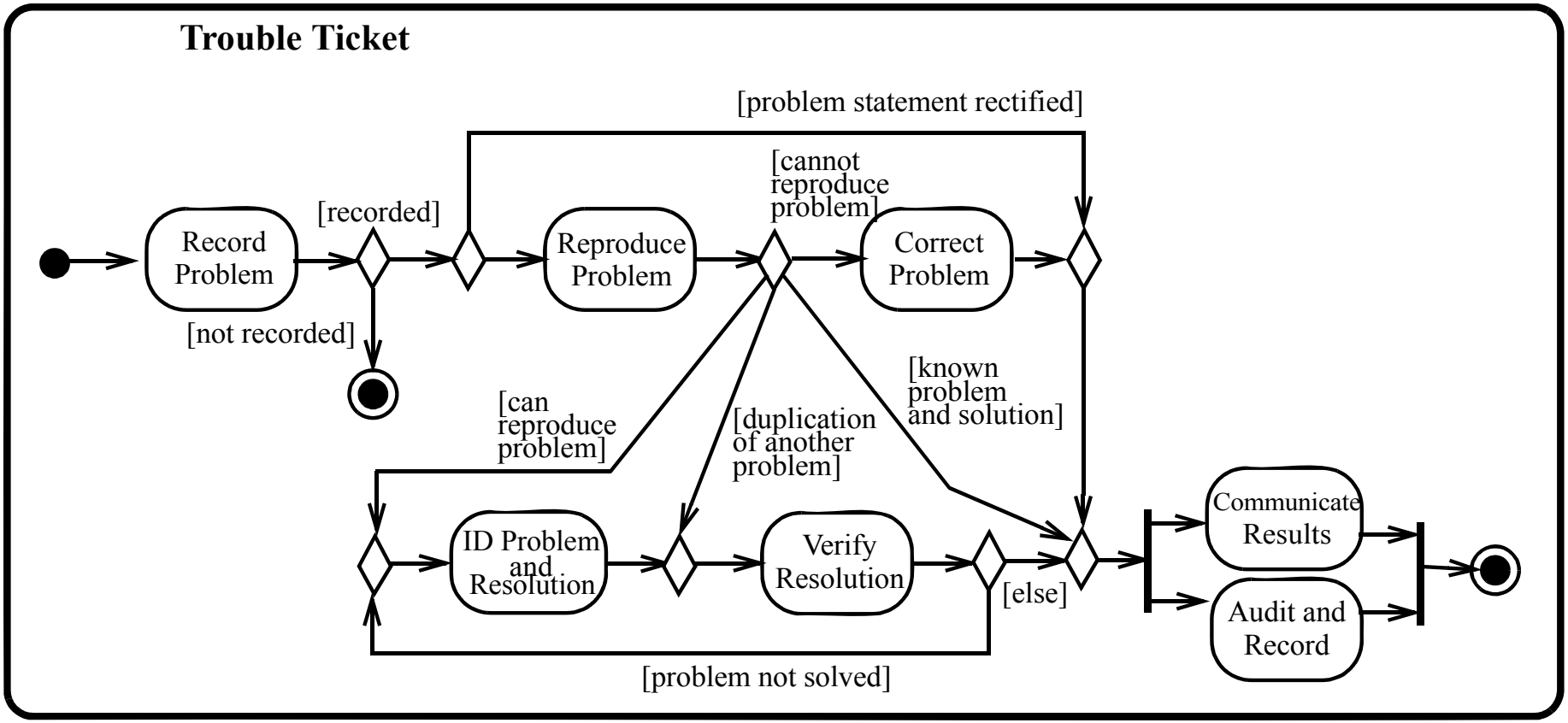
- Ci sono alcuni nodi azione specializzati che gestiscono l'invio e la ricezione di segnali.
- L'invio di segnali è **asincrono** e **non blocca** l'attività.

Segnali ed eventi (2)



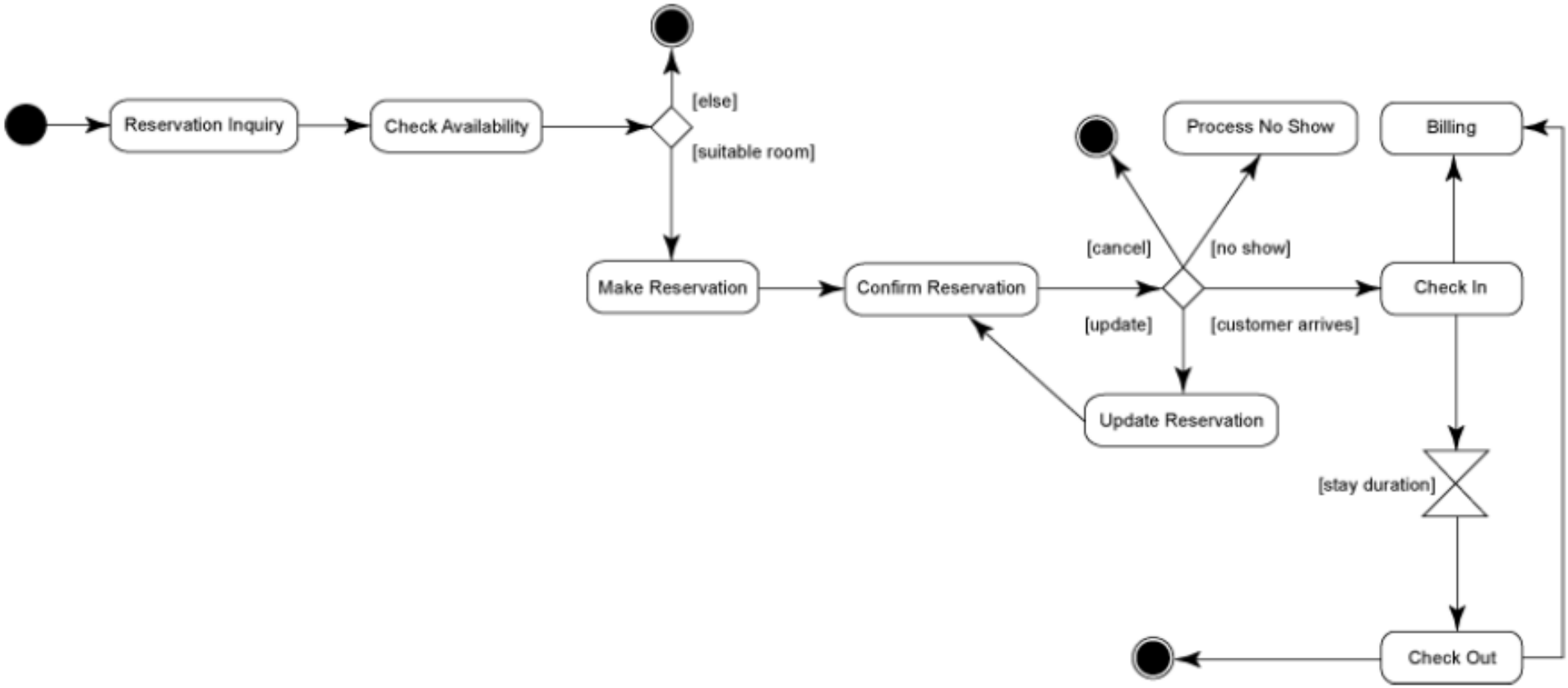
- I nodi ricezione sono attivi quando hanno token su tutti gli archi in entrata (se ne hanno) oppure durante l'intera vita dell'attività (se non ne hanno); generano token alla ricezione.
- La ricezione di eventi temporali funziona nello stesso modo, i token sono generati in base ad un'espressione temporale.

Attività: esempio



Un'attività è costituita da un flusso di azioni che ne sono i mattoni. In effetti un'azione può invocare un'altra attività.

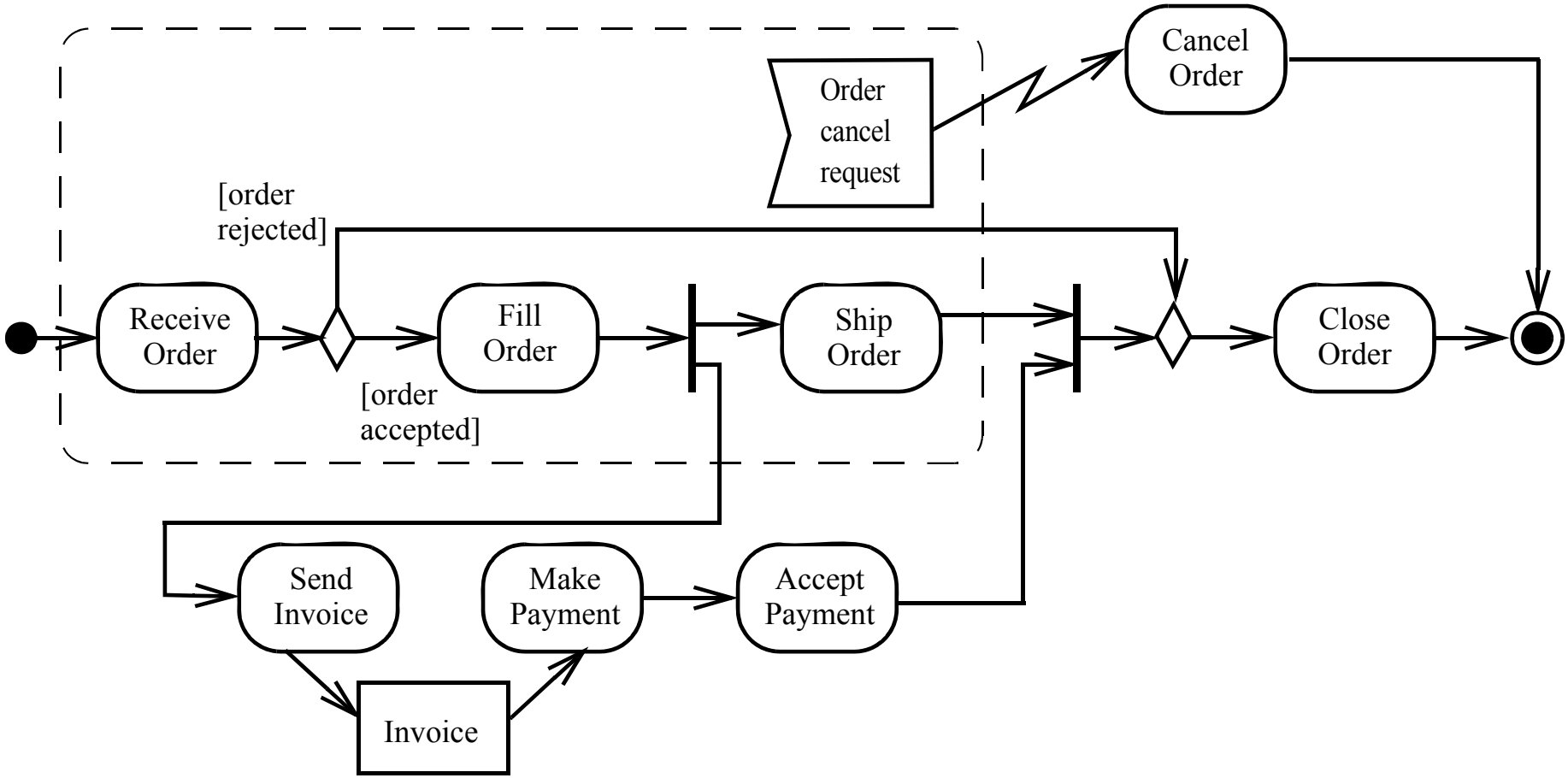
Attività: esempio (2)



[Regioni interrompibili (1)]

- Si usano per specificare una reazione che può avvenire in qualunque momento e comporta l'interruzione dell'attività.
- Esempi: eccezioni, interrupt, segnali, situazioni di errore dall'esterno.
- La notazione impiegata è quella di un'attività con i bordi tratteggiati. Uno o più archi di interrupt (a zigzag) partono da nodi interni e puntano verso nodi esterni.
- L'interrupt è generato quando un arco di interrupt è attraversato da un token: tutti gli altri token e comportamenti nella regione sono terminati.
- La ricezione di eventi all'interno della regione funziona solo se ci sono token al suo interno.

[Regioni interrompibili (2)]



L'ordine è cancellato solo se un token si trova all'interno della regione al momento della ricezione del segnale.

Attività vs. Stato

- In UML 1.x, i due diagrammi sono in pratica la stessa cosa, ma usata in due modi diversi (i diagrammi di attività sono diagrammi di stato in cui gli stati sono azioni).
- UML 2 (qui utilizzato) ridefinisce e separa la semantica dei due diagrammi: quelli di attività si basano sulle reti di Petri, quelli di stato sulla ricerca di Harel.
- Dal punto di vista del modellatore, in UML 1.x entrambi i diagrammi sono diagrammi di stato privi di alcune funzionalità introdotte con UML 2.

Attività vs. Stato (2)

- Uno stato al contrario di un'azione dei diagrammi di attività è solitamente rappresentato con aggettivi e nomi piuttosto che verbi.
- Nei diagrammi di stato non si usano token; le transizioni sono effettuate quando avviene l'evento corrispondente.
- Lo stato iniziale e quello finale si rappresentano allo stesso modo in entrambi i diagrammi (cerchio nero e cerchio bordato).
- Altri elementi di notazione in comune con i diagrammi di attività sono i nodi decisione (decidono lo stato di destinazione in base a una guardia) e i nodi fork/join, che permettono al sistema di trovarsi in vari stati ortogonali (paralleli) allo stesso tempo.

Conclusioni

- I diagrammi di attività descrivono un flusso di azioni che realizzano un certo comportamento specifico. L'enfasi non è sullo scambio di messaggi ma sui blocchi di comportamento.
- I diagrammi di macchina a stati si concentrano su un solo classificatore di contesto e modellano il suo stato interno in relazione al suo comportamento o alle operazioni che possono eseguite sulle sue istanze.
- Come tutti i diagrammi UML, possono essere usati sia a livello di analisi che di progettazione.