Diagramma di macchina a stati

Introduzione

Obiettivi:

- Comprendere il state machine diagram, affinchè possa essere punto di raccordo tra diagramma dei casi d'uso e delle classi

UML include la modellazione di **diagrammi di stato**, principalmente utilizzati per illustrare **eventi** e **transizioni**. Occorre fin da subito sottolineare la duplice distinzione che contraddistingue questo diagramma, in relazione a due **modelli comportamentali**, quali:

- *Behavioral state machine*, descrive un *evento comportamentale* del sistema o di una parte del sistema, come un **attraversamento** dei *vertici* che compongono il chart, connessi mediante *transizioni*
- *Protocol state machine*, descrive il ciclo di vita oppure sequenze di interazioni valide per parti del sistema. Quindi in questo caso si evince la presenza di un *protocollo*, in grado di poter rescindere quali azioni possano essere considerate corrette o sbagliate

Un primo approccio a questa nuova tipologia di chart è dedicata mediante una serie di definizioni informali, che possano garantire una corretta visione di quali strumenti siano necessari pur di poter manipolare questa semantica.

Elementi del diagramma

Solitamente un *state machine diagram* ritrae tutte le entità che compongono la sintassi del diagramma, illustrando eventi e stati di interesse di un oggetto, narrandone il comportamento dello stesso qualora soggetto a stimoli esterni.

Definizione informale
Un **evento** è una occorrenza significativa, similmente, anche se ritrae un esempio scontato e banale, può essere (*Alzare la cornetta del telefono*).

Trovare qualche nozione al riguardo...

Definizione informale
Uno **stato** è la condizione di un oggetto in un dato momento; riprendendo quanto detto prima potrebbe essere indicato uno stato come segue (*Il telefono non termina di squillare fino a quando il ricevitore non alza la cornetta*).

Il concetto su cui si fonda uno *stato* permette di valorizzare il *lifecycle* di un'istanza, come l'esecuzione di un'attività oppure l'attesa posta per la ricezione di un evento specifico.
L'uso degli *stati* avviene per differenziare oggetti tra di loro, dove ognuno dei quali assumerà una condizione differente rispetto a stessi dati immessi. Quest'ultimo passaggio consente di diversificare due tipologie di stato, quali:

- Stato semplice, non ha vertici interni nemmeno transizioni
- Stato composto, contiene una o più sottosequenze, o meglio definiti *sottostati*

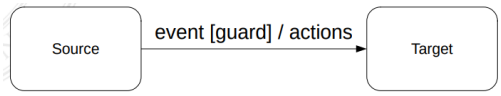
Un'istanza affinchè possa assumere una certa condizione deve essere sottoposta a stimili esterni, i quali comunemente sono denominati **comportamenti**. I *comporamenti* sono artefici del cambiamento di stato di un oggetto, i quali sono riscontrabili in tre differenti fasi, ossia:

- *entry*, è comunicativo qualora siano immesse azioni che stimolino lo stato, per cui producendo un risultato osservabile
- *exit*, avviene nella fase conclusiva, da cui fuoriesce la transizione necessaria che colleghi uno stato al suo successivo
- *doActivity*, è un behavior eseguito assieme ad ulteriori stati associativi, il quale viene interrotto se non conclude la propria esecuzione qualora si esca dallo stato, a causa della transizione

Graficamente uno *stato* è rappresentato mediante un rettangolo con bordi smussati, dove al suo interno possono essere descritti *comportamenti* che si verificano in base allo stato di avanzamento della condizione.

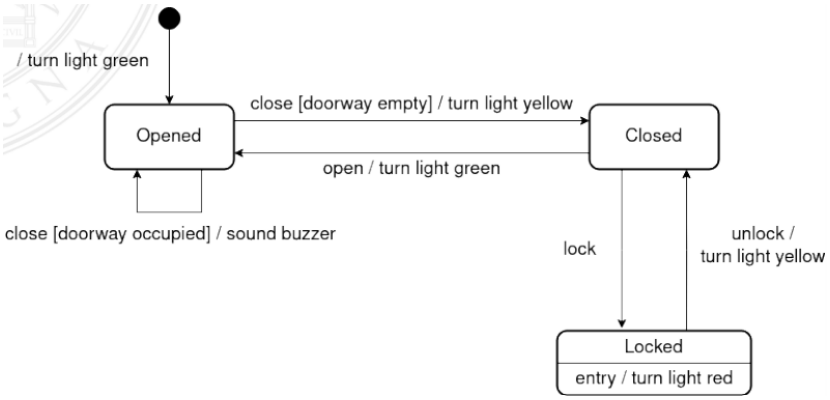
Definizione informale
Una **transizione** è la relazione tra due *stati*, la quale indica qualora un *evento* si concretizzi, il *token* si muove dallo *stato iniziale*, o meglio antecedente, verso lo *stato successivo*. Nuovamente, pur di ottenere un caso reale, si può concludere l'esempio aggiungendo (*Quando l'evento di alzare la cornetta si realizza, il telefono passa dallo stato sonoro allo stato attivo*).

Perciò è definita come il passaggio *atomico* da una *condizione* rispetto alla moltitudine raggiungibile. Al di sopra della *transizioni* possono essere poste delle **guardie**, le quali pongono la possibilità o impossibilità che si concretizzi un evento, contrariamente in assenza, il passaggio dipende unicamente dall'*evento*. Una *guardia* è valutata prima che venga completata la *transizione*.



Rispetto alla notazione introdotta, potrebbero verificarsi casi in cui non sia posta ne una *azione* e nemmeno un *evento*, da cui ne deriva che il passaggio da uno stato precedente ad uno successivo, come potrebbe verificarsi nell'esempio proposto, avvenga senza alcuna interruzione.

Esempio



Nota: tale modello è usato per esprimere un livello di informazioni piuttosto elevato, molto rigoroso, non contribuendo ad alcuna interpretazione possibile.

Si propone una lettura del diagramma in passaggi consecutivi, che possa anche approcciare a concetti teorici, posta come segue:

- Il *nodo totalmente nero* indica il punto di partenza, da cui cominceranno a defluire i differenti token; per certi aspetti simile all'*activity diagram*
- Gli stati sono posti dentro rettangoli con bordi smussati. *Opened* rappresenta lo stato iniziale, inoltre si ricorda la denominazione data, un participio, il quale indica un atteggiamento passivo rispetto all'azione corrisposta
- Le transizioni consistono nei diversi edge, i quali dallo stato iniziale puntano a quello successivo, ma non è escluso che si verifichi esattamente l'opposto. Effettività, pur sempre rapportate allo stato *Opened*, sono *close [doorway occupied]*, la quale indica l'impossibilità nell'apertura della porta, quindi conduce nuovamente allo stesso stato trattato e *close [doorway empty]*, ossia è possibile aprire la porta per poi successivamente chiuderla, ponendo l'evento *close* che varierà lo stato a *Closed*
- Ciò che è posto tra le parentesi quadre, come avviene per *[doorway occupied]*, stabiliscono delle *guardie*, le quali operano in maniera piuttosto selettiva; l'impossibilità che un evento non si verifichi è garantita dalla presenza di tale vincolo, la quale pone un giudizio sulla correttezza del passaggio del token

Un'ultima considerazione aggiuntiva al diagramma prevede che nessuno stato di per sé abbia comportamenti. Uno stato si differenzia da altri corrispettivi poichè prendendo in considerazione *input* differenti corrispondono *output* differenti.

Pseudostati

Definizione informale
I **pseudostati** sono elementi sintattici adoperati per arricchire la semantica di un *state machine diagram.*

Ciò che differenzia uno *stato* da un *pseudostato* consiste nel fatto che un *token* di qualsiasi provenienza, non abbia la possibilità di stagnare permanentemente al suo interno. Un elemento di questo insieme è **initial node**, il nodo da cui iniziano a defluire tutti i *marcatori* all'interno della rete. Infatti, come da esempio precedente, la *transizione* che collega il *nodo iniziale* allo *stato* successivo non possiede una *guardia*; ciò è regolamentato per ogni *diagram* di tale classe, ovviando ad ogni possibile congettura.

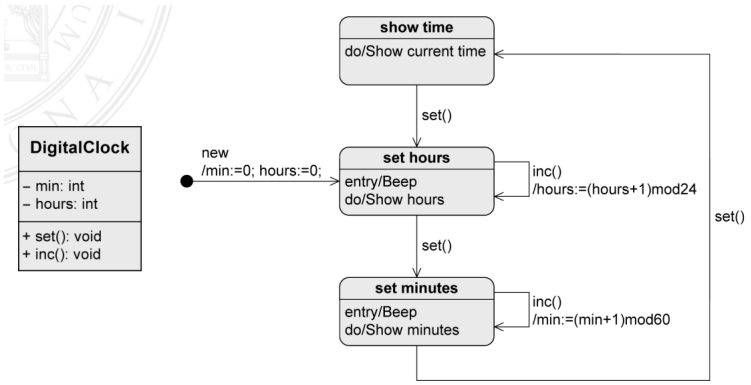
Initial node non identifica l'unico pseudostato, ulteriori sono **join**, **fork** oppure **terminate**; quest'ultimo non deve essere confuso con il **final node**, il proprio obiettivo è totalmente opposto rispetto al secondo citato. Spesso il *terminate* è implementato qualora si voglia improntare un particolare *workflow* alla conclusione, senza che risulti essere vincolante per ogni altro token.

Si elencano usi e rappresentazioni grafiche dei pseudostati:

- Initial, rappresenta il punto di partenza di una *regione*

- Join, simile ad una *swimline* finale, la quale raggruppa differenti transizioni incombi in un'unico edge di uscita. Prima di poter uscire al di fuori della *regione*, ogni funzione deve completare le proprie attività
- Fork, suddivide una transizione in arrivo in differenti *workflow*. Attività contraria rispetto allo *pseudostato join*
- ExitPoint, raccordo di conclusione di una *sottomacchina*
- Terminate, giunto allo pseudostato l'esecuzione della macchina è conclusa immediatamente

Esempio



Un *diagramma di macchina a stati* è adoperato non solo per esprimere nella maniera più dettagliata possibile *eventi* e *transizioni*, ma si presta ottimamente come *design model* per la descrizione di comportamenti per una successiva implementazione.

Per un agevole comprensione è proposto un ulteriore esempio, che raffigura quanto detto prima. Posto il *initial node* la fase successiva, la quale comprende una certa *transizione*, è caratterizzata dalla presenza di un'*azione*, in quanto impone il set dei minuti e ore dell'orologio a zero.

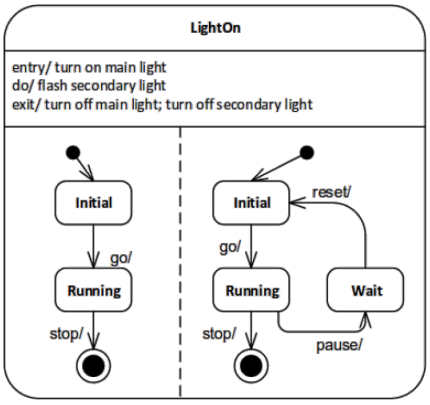
Il primo stato analizzato risulta essere *set hours*, il quale pone due tipologie di *behavior* al suo interno: *entry/Beep*, posto per specificare che all'accensione del dispositivo sarà emesso un suono acuto, e *do/Show hours*, quest'ultimo rappresenta un comportamento di lunga durata, ponendo la visualizzazione dell'orario fino a quando non sia stato raggiunto il valore voluto.

Una nota particolare è data dalla presenza di un'*interazione ciclica*, arricchita dall'azione *inc()*, adoperata per poter modificare il valore associativo alle ore dell'orologio; ciò si riscontra esattamente anche nello stato *set minutes*. Banalmente, i due stati successivi sono perfettamente in linea a quando descritto, ma con alcuni accorgimenti rispetto alla modifica di certi parametri o azioni che debbano conseguire.

Concludendo, l'assenza del *final node* non è un errore, poichè in questo *state machine diagram* i *token* dovranno affluire tra le differenti *transizioni* affinchè sia possibile rispettare la reale volontà nell'uso quotidiano di un orologio.

Regione

Definizione informale
Una **regione** è un insieme di *stati* e *transizioni*, adoperati per rendere maggiormente comprensivo un *diagramma di macchina a stati*. Spesso la sintassi di una *regione* è utilizzata per manipolare contesti che richiedano un quantitativo di elementi piuttosto articolato e complesso, dove una sua assenza renderebbe di difficile analisi il *diagram* ottenuto.

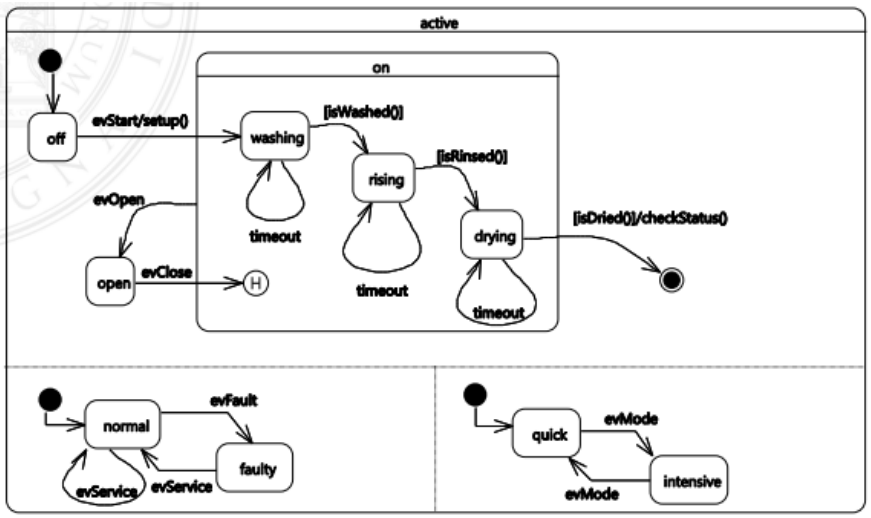


Una regione simile, pone una certa effettività. Un qualsiasi token che si ritrovi all'interno di uno stato simile provvederà ad affluire nelle due **submachine** differenti. Semplicemente, in questa casistica, le *submachine* definiscono comportamenti interni, destinati a terminare, senza influire su *workflow* esterni.

Sintassi aggiuntiva

Definizione informale
Una **compound transitions** rappresenta un insieme di *transizioni* attraversate dal *flow* di *marcatori* generati dal nodo iniziale, le quali non sono vincolate da alcuna *guardia*.

Esempio

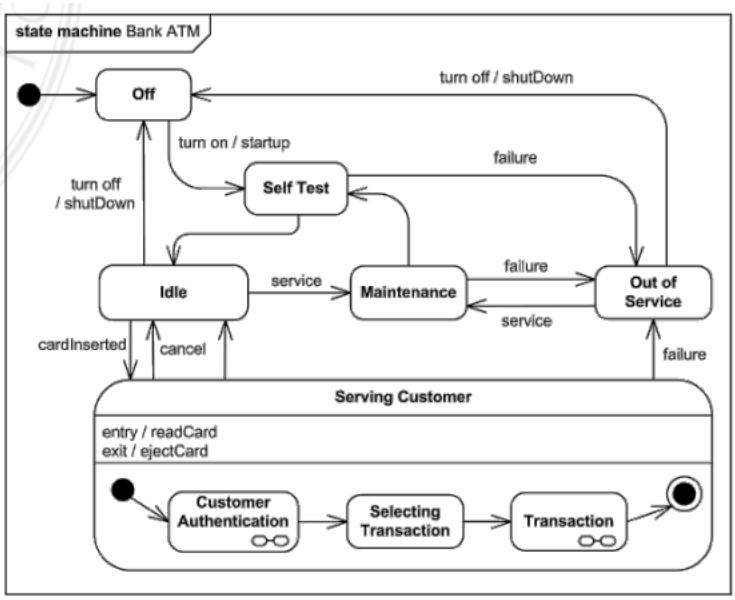


Tale *state machine diagram* raffigura una realtà molto particolare, articolata nell'utilizzo di elementi sintattici piuttosto singolari.

E' rappresentato un unico stato, *active*, da cui si diversificano tre *initial dott* differenti. Solo una *submachine* è caratterizzata da un *flow* destinato a terminare, ossia la sezione comprendente il concreto lavaggio delle stoviglie. La volontà di questa scelta modellativa è dovuta alla continua possibilità, durante l'attivazione della lavastoviglie, di poter modificarne i *pattern* legati alla procedura di lavaggio, come *quick* oppure *normal*.

Si analizza il primo *initial node*. Inizializzato il *token*, il primo step consiste nell'acquisizione dello stato *off*, da cui consegue un azione di *setup*, ossia di preparazione al lavaggio. Descritto ciò, terminata la fase di preparazione, sono elencati tre *stati successivi*, quali *washing*, *rising* e *drying*. La particolarità del *substate* è dovuta all'azione *timeout*; essa, qualora invocata, non provoca una formattazione del processo, ma rimane consistente rispetto all'ultimo stato descrittivo che abbia subito l'azione. Quest'ultimo passo è rispecchiato dal termine sintattico **state history**; mediante tale strumento è possibile tenere traccia dell'attimo in cui si sia verificata un'*interruzione*, garantendo la continuità del *workflow*. Contrariamente, pur sempre in relazione ad un *sottostato*, è possibile resettarne il procedimento.

Esempio



Bank ATM rappresenta un'unica *regione ortogonale*, caratterizzata da un numero piuttosto elevato di *transizioni* tra *stati*. Si osserva la presenza di una singola *submachine*, posta all'interno della condizione *Serving Customer*.

Come avviene per il *state diagram Dishwasher*, i *token* inizializzati defluiscono verso lo stato *Off*, per poi convergere verso *Self Test* eseguendo un'azione istantanea di preparazione, denominata *turn on*. Successivamente, attraversando *Idle* e dopo aver inserito la carta di credito, i *marcatori* sono immessi all'interno della sottomacchina. Lo stato in questione permette di prelevare la somma desiderata; certi participi, come *Customer Authentication* e

Transaction, possiedono un livello informativo molto più dettagliato, grazie alla presenza di ulteriori *state machine* descrittive, indicate dal simbolo sottostante.

Concludendo, dallo stato *Serving Machine*, scaturiscono tre transizioni, ma ognuna di esse è contraddistinta da un comportamento differente. Nella totalità si analizza un'unica transizione che non abbia un'azione o un evento che la caratterizzi, la quale sarà eseguita senza alcun vincolo; mentre, le due corrispettive, sono adottate qualora l'utente decida di eliminare l'operazione oppure in relazione alla casistica in cui si sia verificato un malfunzionamento provocandone il fallimento, con conseguente inattività della macchina.