Come risolvere un problema con una macchina a stati

Scopo di questo articolo è esaminare le modalità di progettazione di una macchina a stati a partire dalla descrizione di un problema di tipo generale. Come punto di partenza si prenderà un esempio classico tratto da manuale di progettazione hardware. Come si vedrà, esiste più di un modo per progettare una macchina a stati per risolvere un problema specifico

Anders Holmberg
IAR Systems

Le macchine a stati sono spesso utilizzate per risolvere determinate tipologie di problemi orientati al controllo dove il flusso di controllo può essere descritto come uno spostamento attraverso un insieme di stati distinti. Esiste un certo numero di definizioni differenti e di descrizioni pratiche di una macchina a stati: a livello teorico si tratta di un automa che risponde a determinati stimoli mediante un cambiamento di stato. La corrispondenza a una specifica espressione regolare (sintassi attraverso le quali si possono rappresentare insiemi di stringhe, ovvero di caratteri) può essere considerata alla stregua di un automa che reagisce alla stringa di ingresso spostandosi attraverso un insieme di stati. Se lo stato prefissato (goal state) viene raggiunto, la stringa di ingresso corrisponde all'espressione regolare. Nel campo del software di controllo è prassi comune basare il modello computazionale della macchina a stati sui concetti di "finito" e di "determinismo": tradotto in pratica ciò significa che il numero dei possibili stati o combinazione degli stati è finito e che la macchina a stati può trovarsi in un solo stato alla volta. Più avanti si vedrà che il concetto di uno stato per volta può essere interpretato in maniera meno rigida, ma l'aspetto principale da considerare è il fatto che lo stato attuale è una proprietà di natura deterministica. Per tutti coloro che hanno poca dimestichezza con le macchine a stati implementate in software, per comodità vengono di seguito riassunti i concetti più importanti.

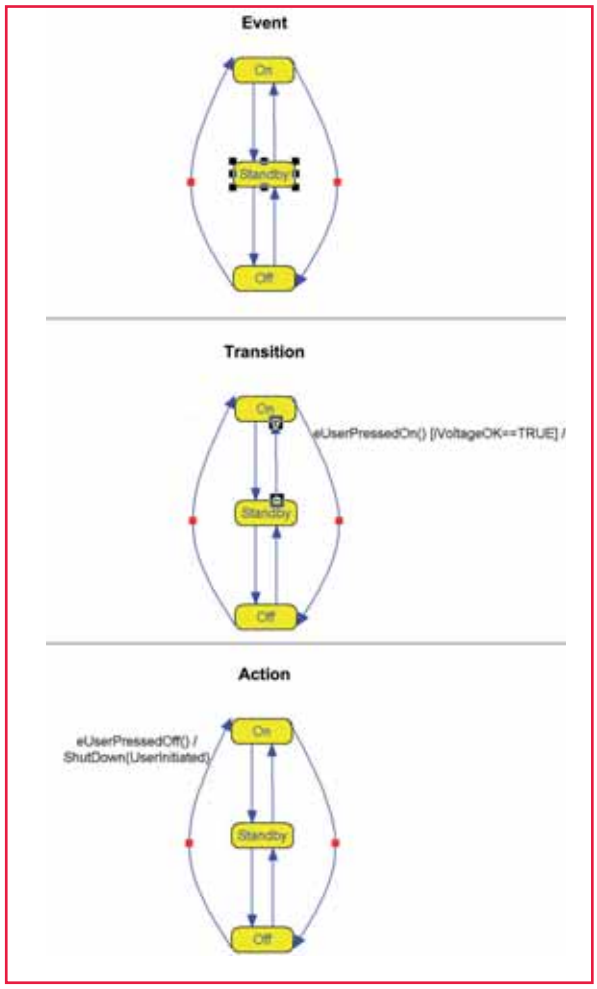

Fig. 1 – Rappresentazione grafica dei concetti di evento, transizione e azione

Stati: uno stato può essere considerato come una descrizione sintetica della situazione attuale del sistema. Come esempio si consideri un dispositivo periferico che può funzionare secondo tre distinte modalità operative: on, off o standby.

Eventi: un evento è un messaggio di ingresso per la macchina a stati. L'evento può provocare un cambiamento di stato della macchina a stati. Ad esempio, la pressione del tasto di "on" di un apparecchio televisivo che si trova in standby può rappresentare un evento che in grado di accendere il televisore, che quindi passa nello stato di on.

Transizioni: una transizione è un percorso diretto da uno stato a un altro. Solitamente la transizione è contrassegnata con il nome di un evento al fine di indicare che la transizione avrà luogo nel caso di verifichi l'evento specifico quando la macchina a stati è nello stato iniziale della transizione. Il risultato finale è rappresentato dal fatto che la macchina a stati cambia stato per raggiungere l'obiettivo (goal state) della transizione. A complicare ulteriormente la situazione – o meglio per incrementare la potenza espressiva della notazione della macchina a stati – una transizione può essere controllata da condizioni di guardia (guard condition) che devono essere soddisfatte affinché la transizione abbia luogo.

Azioni: un'azione è un'attività che deve essere eseguita dalla macchina a stati in un determinato momento. Di solito un'azione espleta alcuni compiti sull'ambiente, come ad esempio la lettura o la scrittura di un dispositivo di I/O. Le azioni possono essere associate alle transizioni, oppure possono essere specificate come azioni di entrata e uscita relativamente a stati specifici. I concetti appena descritti (rappresentati graficamente in Fig. 1) risultano di fondamentale importanza per un approccio formalizzato al progetto di una macchina a stati che si possono trovare come sottoinsieme del linguaggio di modellazione UML.

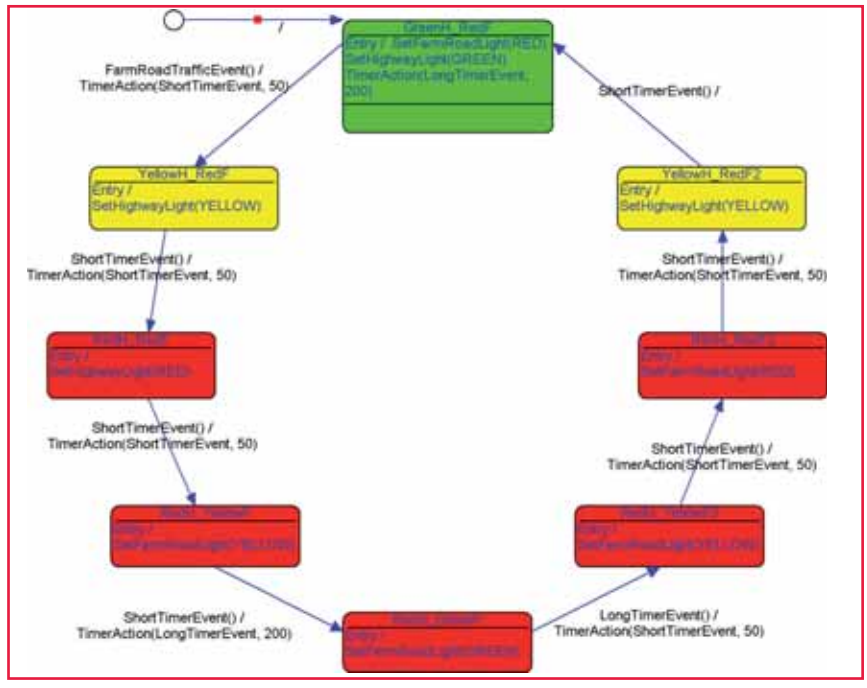

Fig. 2 – Rappresentazione del progetto secondo la simbologia UML

Oltre i concetti base

Di seguito si prenderà in considerazione un esempio tratto da un manuale relativo a un problema orientato allo stato e si cercherà di trovare un'implementazione appropriata mediante una macchina a stati: in realtà in questo caso si definiranno due soluzioni differenti, entrambe caratterizzate da pregi e difetti e, come sempre, la scelta della soluzione più idonea sarà frutto di una valutazione di vari compromessi.

Il problema è il seguente: "Una strada principale molto trafficata è intersecata da una strada secondaria poco frequentata. Appositi rilevatori sono posizionati lungo quest'ultima in modo da far apparire il segnale C non appena un veicolo è in attesa di attraversare la strada principale. Il controllore del semaforo dovrebbe agire nel modo seguente. Se nessun veicolo viene rilevato sulla strada secondaria le luci dovrebbero restare verdi nella direzione della strada principale. Nel caso venga rilevato un veicolo sulla strada di campagna, le luci sulla strada principale dovrebbe passare dal colore giallo al colore rosso in modo da consentire alle luci relative alla strada secondaria di diventare verdi. Le luci della strada secondaria rimangono verdi finché viene rilevato il veicolo su di essa e non devono durare oltre un intervallo prefissato per consentire il flusso del traffico sulla strada principale. Se queste condizioni risultano soddisfatte, le luci della strada secondaria passano in successione da verde a giallo a rosso in modo da consentire alle luci della strada principale di tornare al verde. Anche nel caso di veicoli in attesa di attraversare la strada principale, le luci sulla strada principale devono rimanere verdi per un intervallo di tempo prefissato". [1]

Sebbene un problema di questo tipo sia destinato a un'implementazione di natura hardware, con una certa libertà è possibile adattarlo in modo da renderlo idoneo a una realizzazione in un ambiente software. In questo esempio si adotterà la convenzione che la luce gialla compaia automaticamente da sola. Ciò in realtà non ha effetto alcuno sulla difficoltà del problema, semplicemente facilita l'operazione di tener traccia delle possibili combinazioni di colori.

L'approccio più semplice per progettare macchine a stati può essere descritto nel modo seguente:

1. identificazione degli eventi e delle azioni
2. identificazione degli stati
3. Raggruppamento per gerarchia

4. Raggruppamento per combinazione
5. Aggiunta delle transizione
6. Aggiunta delle sincronizzazioni

Non si tratta ovviamente di una prescrizione rigida, ma di un approccio che può costituire un valido ausilio nel momento in cui sia necessario strutturare una soluzione, indipendentemente dal tipo di tool e dai metodi di implementazione che vengono utilizzati.

Uno sguardo in profondità

A questo punto è necessario esaminare la descrizione del problema andando alla ricerca di possibili eventi e azioni. Prima di iniziare è utile definire cosa si intende per evento nel contesto di una macchina a stati. In sintesi un evento si può considerare una specie di innesco (trigger) che potenzialmente può provocare un cambiamento dello stato di una macchina a stati e l'esecuzione di alcune azioni. Nel caso del semaforo preso in considerazione è possibile trovare un certo numero di potenziali eventi:

- L'arrivo del traffico sulla strada secondaria è un evento significativo per il funzionamento del semaforo.
- Nella descrizione del problema è possibile individuare la presenza di alcuni intervalli di temporizzazione:
  - il tempo minimo durante il quale la luce sull'autostrada rimane verde. Questo può essere utilizzato come tempo fisso per la durata del verde sulla strada secondaria in caso di traffico;
  - il ritardo quando si passa dalla/alla luce gialla su entrambe le strade.

Gli intervalli di ritardo non rappresentano intrinsecamente degli eventi, ma la fine di un intervallo può essere considerato un evento. In effetti esso dà luogo a due eventi: un evento di durata maggiore relativo al periodo durante il quale la luce rimane verde su entrambe le strade e uno inerente al breve periodo (di durata costante) che intercorre durante il passaggio attraverso la sequenza luminosa dal rosso al verde e viceversa.

In definitiva si è in presenza di tre eventi che si potranno definire come FarmRoadTrafficEvent, LongTimeEvent e ShortTimeEvent.

Nel caso del problema che si sta prendendo in considerazione, bisogna considerare ciò che costituisce un'azione. Il cambiamento di colore dei semafori è una tipica azione. La domanda da porsi a questo punto è se il cambiamento della luce su entrambe le strade è da considerarsi come una azione singola oppure come due azioni distinte. In realtà si tratta di una questione di convenienza. Infatti il cambiamento può essere considerato alla stregua di due azioni distinte, ciascuna delle quali si occupa di una strada, oppure essere visto come un singola azione. In questo articolo verrà adottata la prima opzione e la ragione di questa scelta risulterà più chiara in seguito.

Le azioni verranno indicate come SetFarmRoadLight() e SetHighwayLight() e ciascuna azione assume un parametro di colore che indica il colore visualizzato desiderato.

L'idea alla base di ciò è che gli eventi scelti innescheranno cambiamenti in determinate situazioni nel tipo di colori che saranno visualizzati dai semafori.

Inoltre è necessario per poter gestire gli intervalli di temporizzazione; per questo motivo viene introdotta un'azione specifica denominata azione di temporizzazione (timer action) che ha il compito di avviare un timer di conteggio per un intervallo specificato e di seguire un evento che deve essere innescato nel momento in cui il timer ha terminato il conteggio. Questa azione sarà denominata TimerAction().

Per una completa comprensione della logica di controllo del sistema di semafori è necessario un esame approfondito dei possibili stati e delle transizioni tra di essi.

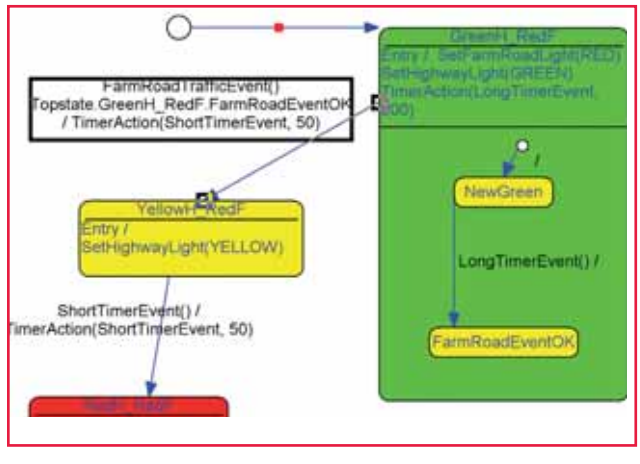

Fig. 3 – Rappresentazione di tre eventi: FarmRoadTrafficEvent, LongTimerEvent e ShortTimerEvent

La prima soluzione è basata sulle seguenti osservazioni:

- In ogni istante di tempo i semafori mostreranno una delle seguenti combinazioni di colore: {VERDE, ROSSO}, {GIALLO, ROSSO}, {ROSSO, ROSSO}, {ROSSO, GIALLO}, {ROSSO, VERDE}, {ROSSO, GIALLO}. In base alle regole di funzionamento previste per i semafori, è chiaro che nessuna altra combinazione è consentita in modo tale da non rischiare che il traffico attraversi l'intersezione da entrambe le provenienze nello stesso momento. Il primo colore di ogni coppia è la luce della strada principale.
- Esiste una sequenza fissa secondo la quale le combinazioni di colore sopra descritta sarà visualizzata: {VERDE, ROSSO}, {GIALLO, ROSSO}, {ROSSO, ROSSO}, {ROSSO, GIALLO}, {ROSSO, VERDE}, {ROSSO, GIALLO}, {ROSSO, ROSSO}, {GIALLO, ROSSO}, con ritorno alla prima combinazione per ricominciare.

Non è inverosimile considerare ogni istanza della sequenza appena descritta come uno stato possibile, in quanto essa rappresenta lo stato effettivo di un sistema in un determinato istante di tempo. Le transizioni da stato a stato saranno provocate dai differenti intervalli di temporizzazione per ciascuna commutazione di colore. Se si ipotizza che la sequenza abbia inizio con la luce verde per la strada principale, la sequenza completa è avviata da un FarmRoadTrafficEvent.

Il lettore perspicace avrà osservato la presenza di alcuni duplicati nelle sequenze di combinazioni di colori. Per esempio la combinazione "entrambe le luci rosse" compare due volte. La domanda da porsi è se debba essere trattata come un singolo stato o come due stati. In questo contesto si è deciso di considerare tutte le posizioni presenti nella sequenza come stati separati. Tale decisione implica che le transizioni tra gli stati risulteranno le più semplici possibili.

In questo momento si dispone delle basi, ovvero di un insieme di eventi, alcune azioni e un insieme di stati candidati.

Nella figura 2 viene riportato il progetto secondo la simbologia UML. Gli stati suggeriti sono stati rappresentati in forma circolare con le transizioni designate dagli eventi corrispondenti. Gli stati hanno anche azioni di ingresso (entry) per cambiare il colore della luce del semaforo. Come si può notare dalla figura, solo lo stato di partenza deve impostare entrambe le luci per assicurare un inizio corretto alla sequenza, mentre per tutti gli altri stati è richiesto il cambiamento di una sola delle luci rispetto allo stato precedente. Questa è la ragione alla base dell'utilizzo di due differenti funzioni di azione. Lo stato iniziale è indicato dalla transizione dallo stato di partenza, contrassegnato dal piccolo cerchio sull'angolo superiore sinistro.

La semantica del linguaggio UML utilizza lo stato iniziale e la transizione a partire da esso per puntare verso lo stato iniziale desiderato: ciò significa che è possibile dichiarare l'inizializzazione del sistema direttamente nel diagramma. La transizione è priva di eventi e implicita all'avviamento.

Dal punto di vista della macchina a stati, il progetto è quasi completo ma non contempla la gestione dell'intervallo di tempo minimo per il verde sulla strada principale. Per questo tipo di problema esistono parecchie soluzioni, anche se quella proposta di seguito risulta abbastanza semplice.

I passi da effettuare sono i seguenti:

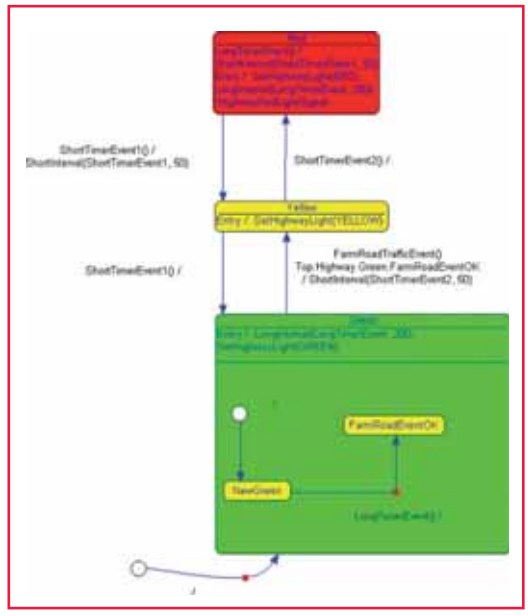

Fig. 4 – Un differente approccio prevede la modellazione delle macchine a stati come due regioni parallele

- Si introduce un certo livello nella gerarchia nel modello mediante la generazione di una macchina a stati all'interno dello stato GreenH_RedF. Questa macchina a stati risulta composta da due stati oltre allo pseudo stato iniziale. Quando lo stato circostante viene inserito, sarà attivato lo stato NewGreen.
- Quando l'intervallo di temporizzazione di lunga durata che era stato avviato dall'introduzione dello stato GreenH-RedF termina, si permette alla piccola macchina a stati di cambiare stato in FarmRoadEventOK. A questo punto è noto che l'intervallo di verde minimo è trascorso per la luce verde sulla strada principale ed è possibile attivare la luce verde sulla strada secondaria in caso di presenza di traffico.
- L'ultima azione che occorre intraprendere per mantenere il minimo intervallo di luce verde sulla strada principale è l'impostazione di una condizione di guardia sulla transizione fuori dallo stato GreenH_RedF. Tale transizione sarà innescata solamente solo se lo stato FarmRoadEventOK è attivato. Ciò può essere espresso dalla condizione di stato positivo o sincronizzazione positiva sulla transizione. Le variazioni risultanti sono rappresentate in figura 3.

La soluzione proposta non è perfetta in quanto non tiene conto del fatto che il traffico in arrivo sulla strada secondaria durante il minimo intervallo di verde sulla strada principale è bloccato finche non arriva un altro veicolo dopo il termine dell'intervallo. Per risolvere questo problema è necessario registrare ogni FarmRoadTrafficEvent che si verifica durante l'intervallo in cui la luce è verde in modo da poter intraprendere provvedimenti adeguati quando termina l'intervallo. In questo caso non si discuterà una soluzione completa del problema, ma nel caso si desideri ideare una soluzione i seguenti punti possono rappresentare validi suggerimenti per una possibile via d'uscita:

- Introduzione di un indicatore (flag) variabile interno alla macchina a stati che viene inizializzato a zero ad ogni ingresso nello stato di luce verde della strada principale.
- Introduzione di una "reazione interna" nello stato di luce verde nella strada principale che imposta l'indicatore nel caso venga ricevuto un FarmRoadTrafficEvent, magari con NewGreen come stato di guardia.
- Una "reazione di ingresso" nello stato FarmRoadEventOK che invia un segnale se il flag è settato quando si entra nello

stato (un segnale è un evento speciale che può essere inviato dalla macchina a stati stessa).
- Una transizione a giallo che innesca il segnale e avvia il temporizzatore dell'intervallo breve.
Esistono altri modi per affrontare questo problema, ma richiedono una sintassi UML troppo complessa per trovare spazio in questo articolo.
Le macchine a stati software differiscono da quelle hardware in quanto una macchina del primo tipo deve rilevare e reagire al verificarsi di eventi nell'ambiente hardware. Ciò significa che il modello computazionale delle macchine a stati software è basato sul ciclo "rilevazione di un evento-reazione all'evento" formato da fasi discrete, mentre una macchina a stati hardware può reagire "simultaneamente" a una variazione di segnale su un determinato pin.
Per esempio un insieme di segnali hardware può essere abbinato in un gate per fornire un evento quando viene soddisfatta una combinazione di segnali in ingresso. In un contesto software è necessario lasciare che la combinazione venga effettuata dai gestori degli interrupt e dai driver dei dispositivi oppure demandare questo compito alla macchina a stati (a meno che non si abbia il controllo del progetto hardware).

Un differente approccio

Come sottolineato all'inizio dell'articolo, esiste almeno una modalità completamente differente di risolvere il medesimo problema che contempla la modellazione delle macchine a stati alla stregua di due regioni parallele nella medesima macchina a stati di livello superiore. Il vantaggio di un approccio di questo tipo è rappresentato dal fatto che è possibile visualizzare la soluzione completa sotto forma di un sistema composto da componenti semi-indipendenti separati (Fig. 4). Per una spiegazione più dettagliata è possibile accedere al sito all'indirizzo: www.iar.com/p236597/p236597_eng.php.
La scelta di una o dell'altra soluzione non riveste una particolare importanza ma è buona norma cercare di adottare l'approccio più semplice possibile. L'utilizzo di un tool di progettazione grafica come visualSTATE di IAR per il progetto di macchine a stati rappresenta un modo semplice per elevare il livello di astrazione, poiché un tool di questo tipo è in grado di supportare funzionalità quali generazione di codice, collaudo e simulazione nonché verifica del modello.

[1] KATZ, Randy H., Contemporary Logic Design. University of California, Berkeley, 1994