Capitolo 10

Le attività di stato

Non si può discendere due volte nel medesimo fiume
e non si può toccare due volte una sostanza mortale nel medesimo stato,
perché a causa dell'impetuosità e della velocità del mutamento
si disperde e si raccoglie, viene e va.
ERACLITO

Esitare va benissimo, se poi si fa quello che si deve fare.
BERTOLT BRECHT

Introduzione

Obiettivo del presente capitolo è terminare l'illustrazione, avviata nel capitolo precedente, relativa ai formalismi forniti dallo UML per modellare il comportamento dinamico del sistema. In quella sede sono stati descritti i diagrammi di sequenza e collaborazione (noti come diagrammi di interazione), mentre in questo capitolo l'attenzione è focalizzata sulle notazioni dei diagrammi delle carte di stato (*statechart diagrams*, indicati anche più semplicemente con il nome di diagrammi di stato) e di attività (*activity diagrams*) che, come si vedrà, rappresentano una variante dei primi.

Sebbene tutte e quattro le notazioni siano dedicate all'illustrazione delle dinamiche del sistema, ognuna di queste permette di modellare gli aspetti dinamici da specifici punti di vista; quindi ognuna risulta particolarmente indicata in determinati contesti e meno in altri. I diagrammi di interazione sono molto utili per modellare il comportamento di un insieme di oggetti che cooperano al fine di raggiungere un determinato risultato, i diagrammi delle carte di stato sono invece utilizzati per mostrare il ciclo di vita di entità quali

istanze di una classe, casi d'uso, il sistema nel suo complesso, ecc. I diagrammi di attività, infine, sono utilizzati principalmente per modellare workflow, ossia flussi di lavoro (processi) rappresentati da un insieme di attività, e dagli oggetti prodotti durante il processo stesso.

Sia la notazione dei diagrammi delle carte di stato, sia quella dei diagrammi di attività, vantano antenati illustri, rispettivamente: gli automi a stati finiti e gli intramontabili flow chart. Questa caratteristica peculiare li rende piuttosto familiari anche a persone con limitate conoscenze tecniche. In particolare, i diagrammi delle attività si prestano a essere utilizzati ogni qualvolta sia necessario modellare un processo, indipendentemente dal settore di applicazione.

Entrambe le notazioni prevedono un utilizzo che abbraccia praticamente tutto il processo di sviluppo del software. Più precisamente, il formalismo dei diagrammi delle carte di stato è adottato, nella fase di analisi dei requisiti, per modellare il ciclo di vita di oggetti business particolarmente importanti; mentre, durante il disegno del sistema, lo si utilizza per rappresentare particolari classi/algoritmi funzionanti con modalità conformi a un automa a stati finiti. Chiaramente, resta valido l'utilizzo atto a descrivere il ciclo di vita di particolari oggetti, anche business, magari aggiungendovi un maggiore dettaglio tecnico.

Per quanto concerne i diagrammi di attività, durante la fase dell'analisi dei requisiti sono impiegati per modellare processi business e scenari dei casi d'uso, ponendo particolare enfasi all'assegnazione delle responsabilità. Nella fase di disegno, sono invece impiegati — sebbene non molto frequentemente — per visualizzare algoritmi particolarmente complessi, enfatizzando eventuali esecuzioni parallele.

I diagrammi di attività costituiscono lo strumento principale utilizzato per modellare i workflow richiesti dai processi di sviluppo del software.

I diagrammi di stato

Introduzione

I diagrammi delle carte di stato — indicati più semplicemente con il nome di **diagrammi di stato** — sono utilizzati per modellare il comportamento delle istanze di un elemento di un modello, come un oggetto o un'interazione. In particolare, sono impiegati per rappresentare la possibile sequenza di stati e azioni attraverso cui le istanze dell'elemento considerato possono transitare durante il proprio ciclo di vita. Ogni transizione, tipicamente, è prodotta dalle operazioni eseguite per reazione agli eventi ricevuti. Esempi di eventi sono un segnale, il trascorrere di un determinato lasso di tempo, l'invocazione di un'operazione, e così via.

L'utilizzo principale del formalismo dei diagrammi di stato consiste nel descrivere il comportamento delle istanze di classi, sebbene non sia infrequente il caso in cui li si utilizzi per modellare il comportamento di altre entità come casi d'uso, attori, sottosistemi, operazioni e metodi.

Nel metamodello UML (fig. 10.5) i diagrammi delle carte di stato sono rappresentati attraverso la metaclasse denominata `StateMachine` che può essere "posseduta" (elemento parte di una relazione di aggregazione) da un classificatore o da una caratteristica dinamica. `StateMachine` eredita dalla metaclasse `ModelElement` la quale è formata, attraverso la relazione denominata `context-behavior`, da altre `StateMachine`. Un opportuno vincolo nel metamodello stabilisce che, se l'aggregazione `context` non è vuota, le classi aggregate — il "tutto" della relazione di aggregazione — devono essere istanze di classificatori o di comportamento dinamico.

Il formalismo dei diagrammi di stato incorporato nelle specifiche dello UML è una rielaborazione OO della notazione dei diagrammi delle carte di stato (*statechart*) elaborata da David Harel, che, a sua volta aveva integrato i modelli delle macchine di stato elaborati da Moor e Mealy.

Definizione accademica

Le basi

Un *automa a stati finiti* (FSA) è il modello più semplice di dispositivo computazionale, formato da un processore centrale di capacità finita basato sul concetto di stato. A sua volta, il FSA si divide in *deterministico* (DFSA) e *non deterministico* (NDFSA).

Un DFSA viene definito come una quintupla

$< Q, I, t, q_0, F >$

dove:

- Q è un insieme finito di stati;
- I è un insieme finito di segnali di input;
- t è la funzione di transizione $t : \{Q \times I\} \rightarrow Q$ che "trasforma" un elemento del dominio (insieme degli input) in un elemento del codominio (insieme degli output). Nel caso in questione l'insieme degli input (dominio) è dato dal prodotto cartesiano dell'insieme degli stati per l'insieme dei segnali di input ($\{Q \times I\}$). Dunque la risposta a un segnale di input dell'automa che si trova in uno stato è un altro stato (non necessariamente diverso dal precedente);
- q_0 è lo stato iniziale, $q_0 \in Q$;
- F è l'insieme degli stati finali, o di accettazione; $F \subseteq Q$.

La funzione $t : \{Q \times I\} \rightarrow Q$ può essere estesa al dominio $\{Q \times I^*\}$, definendo la funzione t^* (ricorsiva) come segue:

$$t^*(q,\lambda) = q \ \forall \ q \in Q$$
$$t^*(q,\chi a) = t(t^*(q,\chi),a) \ \forall \ \chi \in I^* \wedge \forall \ a \in I$$

Ciò significa che è possibile fornire in input all'automa non più soltanto un singolo segnale ma una sequenza di segnali.

Il linguaggio $L(A)$ accettato o riconosciuto dall'automa A è definito come l'insieme di parole accettato da A:

$$L(A) = \{\chi : t^*(q_0,\chi) \in F\}$$

Un NDFSA si differenzia da quello deterministico in quanto la funzione t viene definita nel seguente modo: t è la funzione di transizione, possibilmente parziale (che può non essere definita per alcuni — o tutti — gli argomenti),

$$t : \{Q \times I\} \rightarrow P(Q)$$

dove $P(Q)$ è l'insieme di tutti i sottoinsiemi di Q.

Ciò significa che un NDFSA può eseguire diverse sequenze di transizione per un dato stato e un dato input.

Una stringa di input è accettata da un NDFSA se e solo se almeno una delle possibili sequenze di transizione porta in uno stato finale. I due tipi di FSA sono perfettamente equivalenti, in quanto accettano la stessa classe di linguaggio. Molto spesso però gli NDFSA sono piú comodi da usare perché è più diretto formalizzare un problema in termini di NDFSA.

Teorema 1 (NDFSA è equivalente a DFSA)

"Se L è un linguaggio accettato da un NDFSA, allora esiste un FSA che accetta L".

Il teorema si dimostra costruttivamente. Dato un NDFSA si costruisce un DFSA il cui insieme di stati finali è l'insieme dei sottoinsiemi che contiene almeno uno stato finale del NDFSA di partenza. La funzione di transizione è definita applicando la t del NDFSA a ciascuno stato in un dato sottoinsieme e prendendo l'unione degli stati risultanti.

> Una guida indispensabile alla teoria degli automi e legami con le grammatiche si trova in [BIB36]; a [BIB35] ci si riferisce come *al testo* sugli automi, mentre nel testo [BIB34] è riportata una succinta guida, in lingua inglese, alle basi della teoria degli automi, con link a alcuni semplici esercizi e un interessante programma (sorgenti su richiesta, eseguibile per HP700 con X11-R5 scaricabile).

Tabella 10.1 — *Elenco degli stati e dei segnali dell'automa a stati finiti relativo al lettore CD.*

STATO	SEGNALE
OFF	off
INIT	on
!DISK (STOP)	stop
DISK (STOP)	play
TRAY OPEN	pause
PLAY on OPEN	eject
PLAY	
PAUSE	*!disk*
OFF on OPEN	*disk*
EJECT on PLAY	*ioff*
OFF on PLAY	*ieject*

Esempio

Di seguito viene proposto un interessante esempio di un automa a stati finiti: il lettore di compact disc. Considerati i fini della presente trattazione, si tratta di un esempio semplificato, ma non troppo. L'automa che lo modella è costituito da undici stati, e da dieci segnali di lavoro (sei "volontari" e quattro "involontari", ovvero risposte dell'apparecchio). Degli stati elencati, quelli in corsivo sono stati "fittizi" introdotti artificialmente per evitare di rappresentare le funzioni di transizione *starred* ("ricorsive"), avendo quindi un'esplosione in funzioni non ricorsive di più immediata comprensione. Allo stesso modo i segnali in corsivo sono segnali "involontari" che il sistema è in grado di fornire a sé stesso per eseguire un'azione obbligata (tab. 10.1).

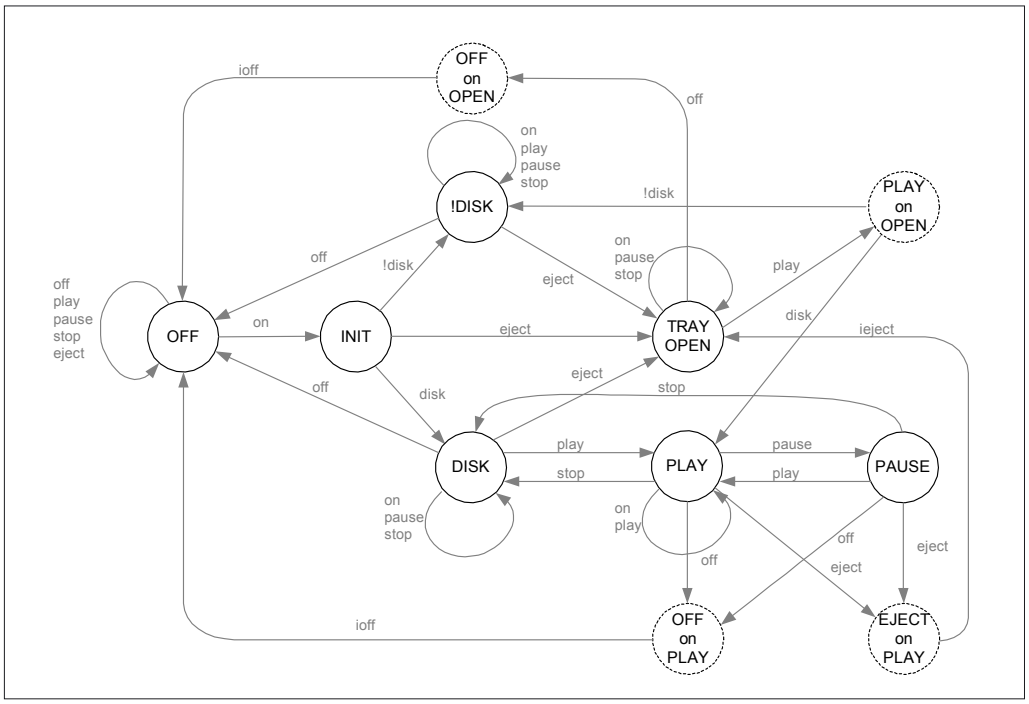
Si consideri per esempio la funzione *starred* per vedere qual è il principio usato per semplificare l'automa in esame. Si supponga che l'automa sia nello stato `OFF` e che riceva il segnale `on`. Il lettore di CD si accende, esegue il controllo sulla presenza di un disco e transita nello stato di `STOP`. In funzione del risultato della verifica, il lettore può visualizzare una rappresentazione della ToC (*Table of Contents*, tabella dei contenuti) del disco, oppure il messaggio *No Disc*. Il segnale ricevuto è quindi `on*`, composto (da `on` e `checkdisc`) e la funzione di transizione è

$$t^*(OFF, on^*) = t(t(OFF, on), checkdisc) = t(t(t^*(OFF, \lambda), on), checkdisc)$$

dove

$t^*(OFF, \lambda) \rightarrow OFF$ per definizione.

Figura 10.1 — *Automa a stati finiti relativo a un lettore di CD.*



Rimane da processare `t(t(OFF,on),checkdisc).t(OFF,on)` → `INIT`, nome di comodo di uno stato di inizializzazione. In questo stato, definito fittizio, si esegue il controllo sulla presenza del disco (segnale `checkdisc`).

È abbastanza chiaro che tale sottoautoma è un NDFSA, dato che, a seconda della risposta all'unico segnale `checkdisc`, si transita in uno stato (si mostra la ToC) o in un altro (si mostra *No Disc*).

Nell'automa, che deve essere deterministico, è stata simulata questa funzione con due segnali "involontari" (`disk` e `!disk`) che portano in due diversi stati (`DISK` e `!DISK`), entrambi di `STOP`. Lo stesso ragionamento è applicabile nelle situazioni analoghe che si incontrano nel caso in oggetto. Si vedrà di seguito coma la notazione dello UML, grazie soprattutto al meccanismo delle transizioni interne, permette di evitare molti di questi artifici.

Si ricordi che il dominio è rappresentato da tutte le possibili coppie di Stato-Segnale, mentre il codominio è l'insieme degli stati.

Nell'elenco seguente sono riportate le regole di produzione della funzione di transizione solo per lo stato `OFF` (gli altri sono immediatamente deducibili dall'automa di fig. 10.1, come utile esercizio per il lettore):

```
t(OFF, off)    → OFF
t(OFF, on)     → INIT
t(OFF, stop)   → OFF
t(OFF, play)   → OFF
t(OFF, pause)  → OFF
t(OFF, eject)  → OFF
```

Molti lettori CD prevedono transizioni dallo stato `OFF` diverse da quelle mostrate nell'elenco precedente. Per esempio, premendo il tasto play a lettore è spento, avviene tutta una sequenza di azioni: il lettore si accende, effettua il controllo di presenza del disco e, in caso di esito positivo, inizia a suonare il primo brano. In questo caso si è preferito semplificare il comportamento al fine di contenere la complessità dell'automa.

Notazione UML

Stati e transizioni interne

Da quanto riportato nel paragrafo precedente segue che la notazione dei diagrammi delle carte di stato è utilizzata prevalentemente per rappresentare automi a stati finiti, con diverse differenze. In particolare, trattandosi di un formalismo più recente adattato al paradigma OO e in continua evoluzione, è ovviamente più potente, come si avrà modo di notare negli esempi successivi.

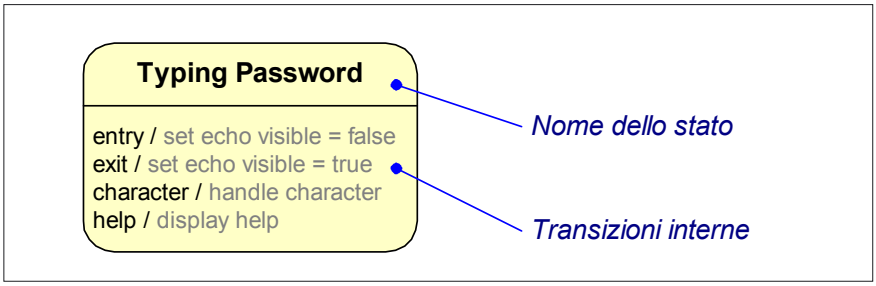
Dal punto di vista della notazione, la definizione generale dice che un diagramma è un grafo costituito da vertici e archi che li connettono. In questo contesto, i vertici del grafo sono gli stati (e altri elementi similari come gli pseudostati); le connessioni tra questi rappresentano tipicamente le transizioni di stato.

Uno stato rappresenta una condizione durante il ciclo di vita di un oggetto, o di un'interazione, durante la quale l'oggetto soddisfa determinate condizioni, esegue specifiche operazioni e attende l'occorrenza di opportuni eventi.

Ogni diagramma delle carte di stato possiede, per definizione, uno stato di livello superiore, il quale contiene tutti gli elementi del diagramma delle carte di stato considerato. Tipicamente questo stato superiore non è visualizzato, e lo UML non definisce alcuna particolare notazione grafica atta a visualizzare la connessione tra uno stato e il suo contenuto. Quella tipicamente utilizzata prevede la visualizzazione classica dello stato superiore con la macchina a stati dettagliata all'interno (fig. 10.3).

> Nel metamodello UML (fig. 10.5) l'elemento `StateMachine` (che rappresenta il concetto dei diagrammi di stato) è costituito da un solo stato, top-level. Ciò è rappresentato dalla relazione di composizione con la metaclasse `State`, la quale prevede una serie di elementi specializzanti, tra cui `CompositeState`, ossia uno stato costituito da altri stati (rappresentato mediante una relazione di composizione con l'elemento `StateVertex`, genitore di `State`).

Figura 10.2 — *Esempio di uno stato atto a acquisire la password da tastiera. Esempio tratto dalle specifiche ufficiali (fig. 3-73).*



Gli stati sono mostrati graficamente attraverso rettangoli dagli angoli smussati (fig. 10.2), eventualmente suddivisi in diversi compartimenti. Se necessario è possibile specificare un'apposita "linguetta" (*name tab*) atta a mostrare il nome dello stato. Questa linguetta viene attaccata al lato superiore del rettangolo dello stato e si dimostra (come si vedrà a breve) particolarmente utile per mostrare il nome di stati composti aventi regioni concorrenti (fig. 10.7).

Il rettangolo che rappresenta lo stato può essere suddiviso attraverso opportune linee orizzontali in una serie di compartimenti (fig. 10.2). Il primo, opzionale, è utilizzato per rappresentare il nome dello stato. Uno stato senza compartimento relativo al nome è detto anonimo.

Qualora nello stesso diagramma siano rappresentati diversi stati anonimi, questi vanno considerati distinti. Al fine di evitare confusioni è sconsigliato mostrare, in uno stesso diagramma, più stati con lo stesso nome o più stati anonimi.

Nei casi in cui sia mostrata l'etichetta del nome (*name tab*) menzionata prima, il compartimento del nome va evitato e viceversa (si tratta di notazioni alternative).

Il secondo compartimento è relativo alle transizioni interne (*internal transitions*), e è utilizzato per mostrare una lista azioni o attività interne da eseguire quando l'istanza si trova in tale stato e si verifichino particolari eventi (entrata, uscita, ecc). Per esempio, nel diagramma di fig. 10.2 sono mostrate le transizioni interne: `entry`, `exit`, `character`, `help`. Da notare che solo le prime due sono standard, mentre le rimanenti due sono state definite dal modellatore.

Il nome di questo comportamento è dovuto al fatto che le transizioni interne sono del tutto equivalenti alle autotransizioni (*self transition*) eccetto che per la "non terminazione" dello stato, e quindi per la mancata uscita e il mancato rientro dallo stesso. Ciò comporta che non si innescano eventuali azioni associate a tali eventi. Come si vedrà a breve (figg. 10.3 e 10.4), questo meccanismo permette di rappresentare in maniera relativamente semplice macchine a stati finiti anche molto complesse.

Il formato di una transizione interna prevede la seguente configurazione:

```
event-name '(' comma-separated-parameter-list ')' '[' guard-condition ']' '/'
action-expression
```

I nomi degli eventi sono mostrati attraverso opportune etichette il cui compito è identificare le circostanze che determinano l'esecuzione delle relative azioni (entrata, uscita, durante la permanenza, ecc.). Le azioni sono mostrate attraverso appropriate espressioni (*action-expression*), nelle quali è possibile utilizzare attributi e collegamenti appartenenti alla visibilità dell'entità oggetto della modellazione.

È possibile inoltre specificare delle condizioni di guardia (*guard-condition*) il cui risultato booleano della valutazione determina l'esecuzione o meno delle relative azioni.

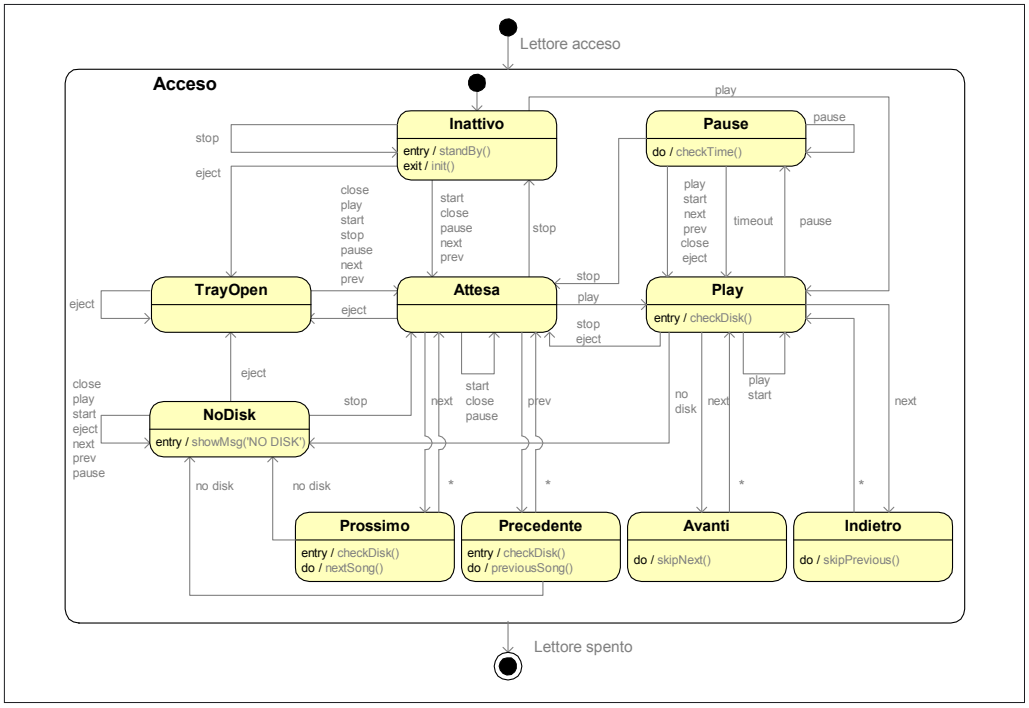
Uno stesso nome evento può comparire diverse volte nell'ambito del medesimo stato, a condizione che l'esecuzione delle relative azioni siano soggette a diverse condizioni di guardia. Le etichette sono separate dalle azioni attraverso opportune barre oblique (/), quelle che in lingua inglese, e ormai anche nell'uso comune, sono dette *slash*.

Lo UML predefinisce le etichette riportate nella tab. 10.2 che pertanto non possono essere utilizzate per nomi di eventi.

Tabella 10.2 — *Nomi eventi predefiniti in UML corredati dai relativi significati.*

Nome	Descrizione
entry *Entrata*	Identifica un'azione la cui espressione deve essere eseguita All'atto dell'entrata nello stato.
exit *Uscita*	Identifica un'azione la cui espressione deve essere eseguita all'atto dell'uscita dallo stato.
Do *Esegui*	Si tratta di un'etichetta che identifica un'azione da svolgere mentre l'elemento modellato si trova nello stato (in questo caso identifica un'azione in esecuzione) o fintantoché la relativa espressione non viene completata.
Include *Inclusione*	Questa etichetta è utilizzata per identificare un'invocazione di una submachine. In questo caso l'espressione deve contenere il nome della submachine da eseguire.

Figura 10.3 — *Rappresentazione UML dell'automa a stati finiti relativo al lettore CD dotato di funzioni di selezione brani e avanzamento traccia. Il simbolo "asterisco" indica la presenza di tutti gli stimoli, e è stato introdotto per esigenze di chiarezza espositiva. La presenza delle transizioni interne ha permesso di modellare l'automa in modo meno complesso. In particolare non è stato necessario ricorrere a tutta una serie di "stati artificiali".*



Nelle figg. 10.3 e 10.4 sono mostrati due diagrammi relativi, rispettivamente, al ciclo di vita del lettore CD e del bancomat, che fanno notevole uso delle transizioni interne.

> La descrizione del comportamento dinamico del diagramma di fig 10.4 è demandata al lettore, per evidenziare la semplicità di fruizione dei modelli rappresentati attraverso il formalismo dei diagrammi degli stati.

Gli stati possono essere di tre tipi fondamentali: stato semplice, stato finale e stato composto. Nel metamodello (fig. 10.5) sono rappresentati mediante gli elementi `SimpleState`, `FinalState` e `CompositeState` che ereditano dalla metaclasse `State`. Lo stato composto è poi ulteriormente specializzato nello stato di sottomacchina (`SubmachineState`).

Figura 10.4 — *Diagramma degli stati utilizzato per modellare il ciclo di vita di un bancomat.*

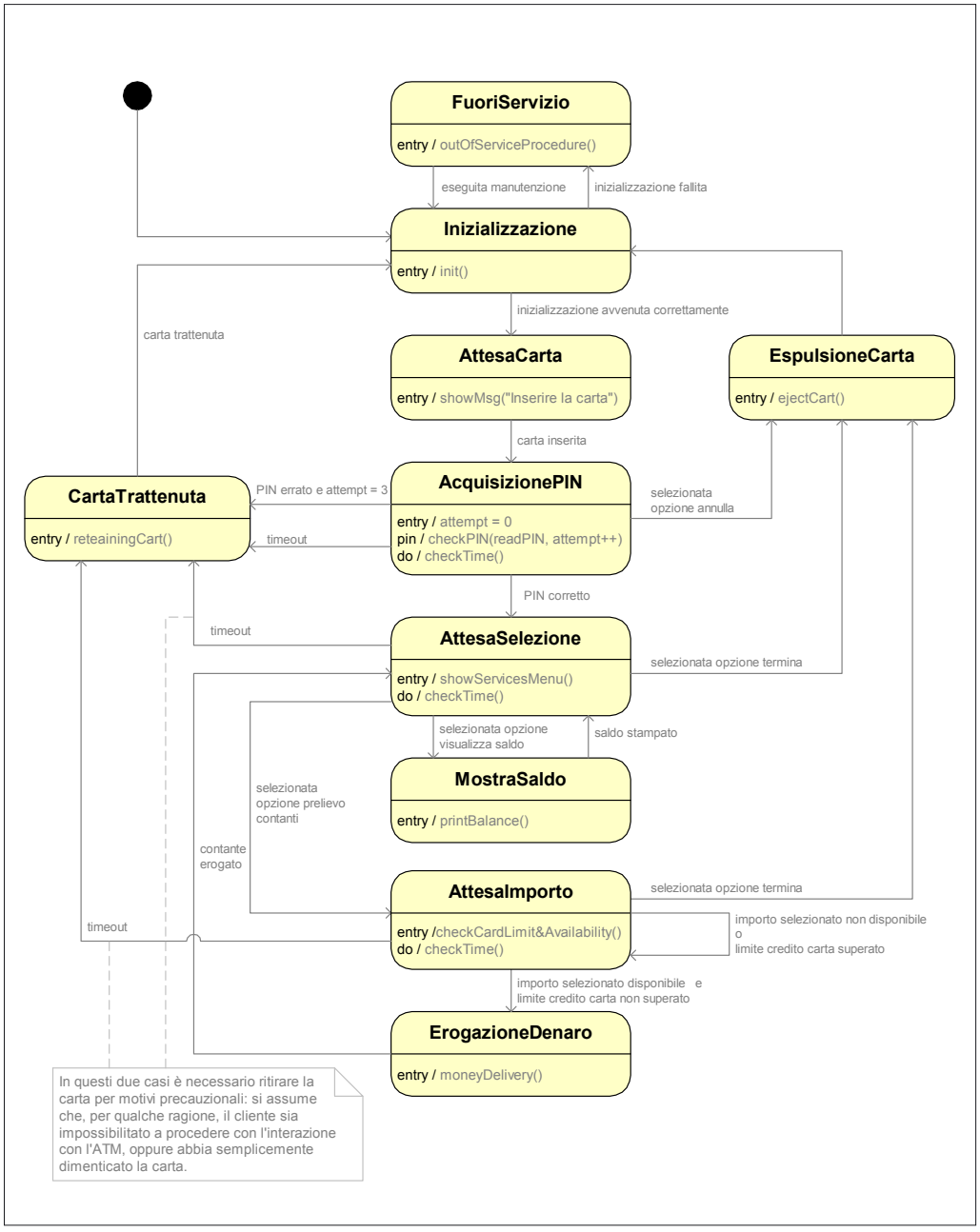
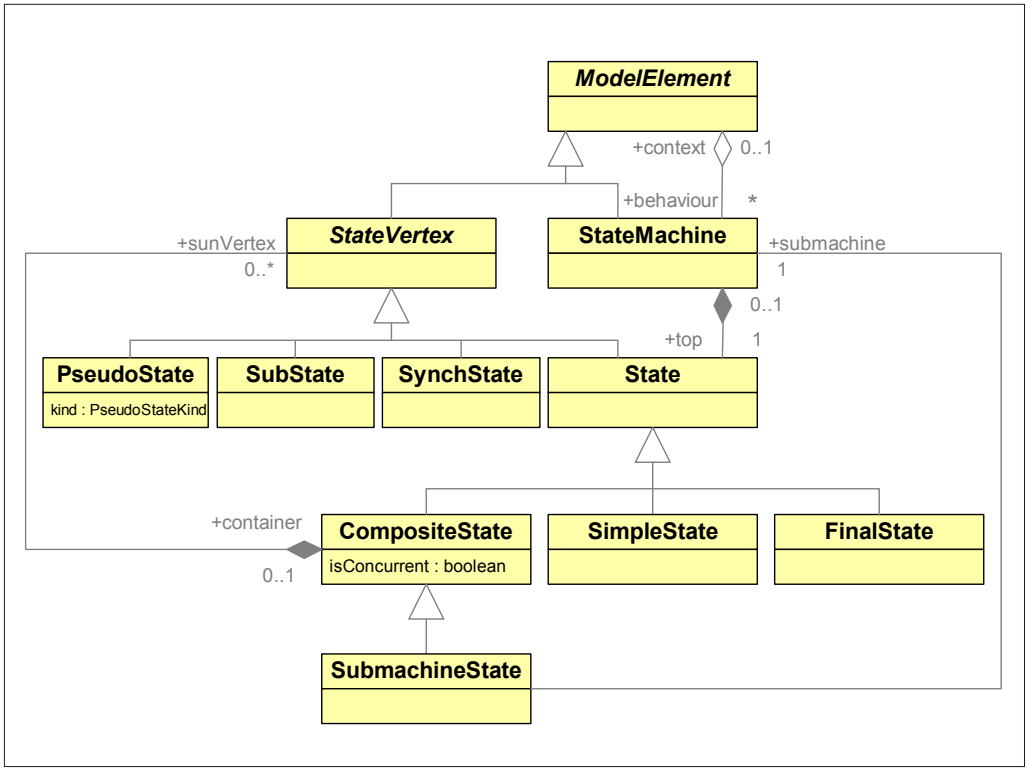


Figura 10.5 — *Rappresentazione semplificata della porzione del metamodello relativa alla gerarchia degli stati dell'elemento* `StateMachine`. *L'attributo* `isConcurrent` *della metaclasse* `CompositeState` *è utilizzato per distinguere stati composti, formati da sottostati concorrenti (regioni), da quelli costituiti da sottostati mutuamente disgiunti. Mentre l'attributo* `kind` *della metaclasse* `PseudoState` *serve per indicare la tipologia del pseudostato: iniziale, di diramazione, di ricongiunzione, storico, ecc.*



Come è lecito attendersi, uno stato semplice è caratterizzato dal non prevedere un'ulteriore decomposizione e pertanto è antitetico allo stato composto che invece la prevede.

Tra i vari pseudostati, uno di particolare importanza è quello iniziale, il quale indica la creazione dell'istanza dell'elemento di cui si modella il ciclo di vita, e quindi identifica l'avvio della macchina a stati. Come da standard UML, lo **pseudostato iniziale** è rappresentato attraverso una piccola circonferenza piena (figg. 10.3 e 10.4). La transizione da uno pseudostato iniziale può essere etichettata con l'evento che genera l'oggetto, di cui si rappresenta il ciclo di vita, mentre non deve essere riportata nelle transizioni da stati

iniziali di sottostati composti. In questi casi, l'evento è fornito da una qualsiasi transizione che termina nello stato composto. Da notare che è possibile associare un'azione alla transizione iniziale. Uno pseudostato iniziale può avere, per definizione, una sola transizione uscente e nessuna entrante.

Lo **stato finale** indica che lo stato composto che lo ospita ha terminato il proprio ciclo di vita. Qualora questo stato sia quello di primo livello, significa che l'esecuzione dell'intera macchina a stati è terminata. Lo stato finale è rappresentato attraverso due piccole circonferenze concentriche, di cui quella più interna piena (la classica configurazione denominata a "occhio di bue", fig. 10.3). Qualora lo stato finale appartiene a uno stato composto, il relativo raggiungimento determina la transizione di uscita dello stato stesso. Ciò innesca l'esecuzione dell'eventuale azione di uscita dello stato composto, nonché la relativa transizione inerente al completamento delle attività interne.

> Nel metamodello UML, lo stato finale è rappresentato per mezzo di un'apposita metaclasse (fig. 10.5) e è caratterizzato dal non possedere transizioni uscenti (`FinalState.outgoing -> size = 0`).

Concettualmente uno stato è immaginato come una condizione in cui l'oggetto permane per un certo lasso temporale. Si tratta indubbiamente della situazione più frequente. In alcuni contesti però è necessario ricorrere a particolari stati denominati di "transizione". Come suggerisce il nome, si tratta di stati di passaggio in cui non è prevista la permanenza. Chiaramente la notazione dello UML permette di modellare stati sia istantanei che non. Gli stati poi possono essere utilizzati per modellare attività in corso di esecuzione rappresentata attraverso un opportuno diagramma delle carte di stato annidato o per mezzo di un'espressione computazionale.

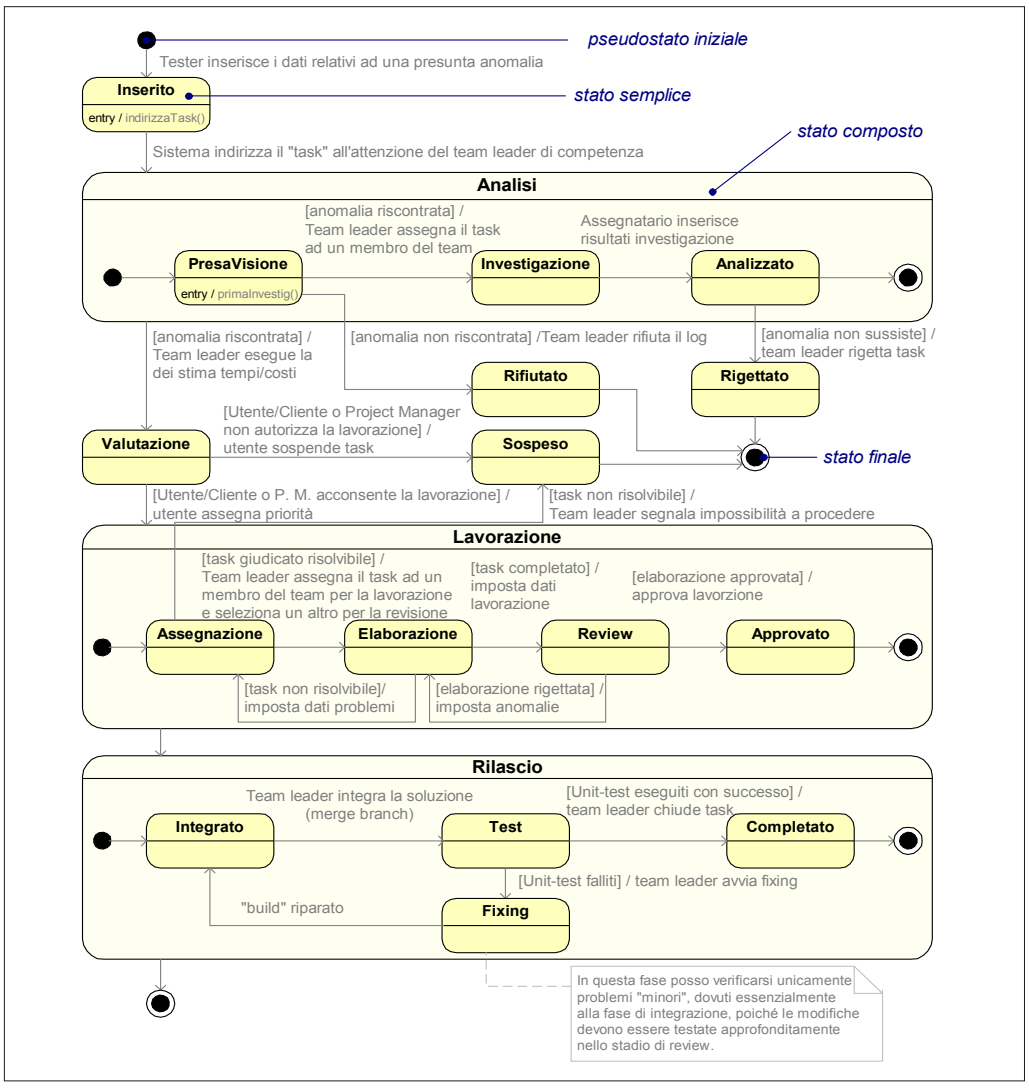
Stato composto

Uno stato è detto composto quando può essere ulteriormente definito per mezzo di una macchina a stati. Esistono due tipologie di stati composti: quella che prevede la relativa definizione per mezzo di due o più macchine a stati concorrenti e quella in cui vi sia semplicemente un insieme di stati connessi.

La definizione formale sancisce che uno stato composto è uno stato che può essere scomposto in due o più sottostati concorrenti (denominati regioni, *regions*), oppure in due o più stati disgiunti mutuamente esclusivi. Ciascun sottostato di uno stato composto può essere a sua volta uno stato composto e può avere propri pseudostati iniziali e stati finali.

La notazione grafica standard degli stati composti prevede un'ulteriore sezione dedicata all'illustrazione della struttura della macchina a stati interna. Per questioni di comodità, spesso il compartimento del nome viene ridotto esclusivamente a un'etichetta riportante il nome che, tipicamente, è collocata nell'angolo in alto a sinistra.

Figura 10.6 — *Ciclo di vita delle richieste di riparazione delle anomalie — vere o presunte — emerse durante la fase di test del sistema. Le richieste possono essere specificate sia dal team di test, sia da un gruppo selezionato di clienti. Le anomalie appartengono tipicamente a due categorie: errori veri e propri (NCR, Non Conformity Requirement, mancata conformità ai requisiti) e richiesta di cambiamento dei requisiti (CRR, Change Requirements Request). Il diagramma degli stati di fig. 10.11 potrebbe essere utilizzato per creare un opportuno software per la gestione di richieste di modifiche da apportare al sistema. Si tratterebbe di un tool estremamente utile di supporto all'attività di gestione dei cambiamenti (Changes Management).*



Stato composto non concorrente

Come visto, uno stato composto non concorrente è uno stato la cui logica interna è dettagliata per mezzo di un'apposita macchina a stati. Consistentemente con quanto riportato fino a ora, l'evoluzione delle relative istanze, inizia dallo pseudostato iniziale o meglio, dalla sua transizione uscente. Quando poi l'oggetto raggiunge lo stato finale, termina il proprio ciclo di vita.

Una transizione che termina in uno stato composto (graficamente la freccia termina sul bordo del corrispondente elemento) equivale a una transizione diretta verso il relativo pseudostato iniziale, il quale deve essere esplicitamente specificato. Poiché si tratta di una transizione proveniente dall'esterno, genera l'innesco dell'evento di entrata nello stato e quindi viene eseguita l'eventuale azione associata. Anche le transizioni dirette a uno stato interno, eventualmente annidato, appartenente a uno stato composto, generano l'esecuzione di tutte le azioni associate all'entrata degli stati attraversati.

Una transizione che termina allo stato finale di uno stato composto non concorrente ne determina il completamento, e quindi innesca l'evento di uscita dallo stato.

Qualora sia innescata una transizione uscente dallo stato composto (l'origine è localizzata sul bordo del corrispondente elemento), relativa al verificarsi di un particolare evento (per esempio un timeout), viene forzata la terminazione delle attività in corso e si eseguono le eventuali azioni associate all'evento di uscita. Alla fine della relativa esecuzione transita nel nuovo stato.

Particolari transizioni di uscita di uno stato composto possono essere tracciate a partire da uno suo stato interno a qualsiasi livello di annidamento.

Si consideri per esempio il diagramma di fig. 10.7. Quando una transizione giunge allo stato composto denominato `Analisi`, la relativa evoluzione inizia dallo stato denominato `PresaVisione`. Se poi l'evoluzione interna si svolge completamente fino a giungere allo stato finale, (dallo stato interno `Analizzato`, il team leader considera fondato il difetto, e quindi ne effettua la stima tempi/costi), si esce dallo stato composto e viene innescata la transizione verso lo stato denominato `Valutazione`.

Stato composto concorrente

Qualora uno stato composto sia costituito da sottostati concorrenti, questi sono mostrati suddividendo la sezione dedicata alla macchina a stati interni in diverse zone orizzontali denominate regioni (fig. 10.7). Tale suddivisione è visualizzata attraverso opportune linee tratteggiate. Ciascuna delle regioni rappresenta un sottostato concorrente, e come tale, può disporre di un nome e può contenere un opportuno diagramma degli stati annidato costituito da stati disgiunti. In sostanza, ogni regione contiene una macchina a stati. In questo caso, qualora si voglia riportare il nome dell'intero stato, questo deve essere visualizzato all'interno del relativo compartimento o per mezzo dell'apposita linguetta. Una transizione diretta verso uno stato composto, dotato di sottostati concorrenti, corrisponde a innescare le transizioni uscenti da tutti gli pseudostati iniziali presenti nelle

varie regioni. Il completamento dello stato è raggiunto a seguito del completamento delle attività in tutte le regioni concorrenti. Questo evento innesca l'esecuzione dell'eventuale azione di uscita specificata nello stato composto contenente le varie regioni.

Una transizione diretta a uno stato di una regione concorrente genera sia l'esecuzione di tutte le azioni associate all'entrata degli stati attraversati, sia l'innesco delle transizioni di uscita dagli pseudostati iniziali presenti nelle restanti regioni.

Particolari transizioni di uscita di uno stato composto possono essere tracciate a partire da un suo stato interno a qualsiasi livello di annidamento. Queste generano l'esecuzione delle azioni di uscita di tutti gli stati da cui si forza l'uscita.

Una transizione uscente dallo stato composto (l'origine è localizzata sul bordo del corrispondente elemento), relativa al verificarsi di un particolare evento (per esempio un timeout), forza la terminazione di tutte le regioni di cui si compone, le quali eseguono le eventuali azioni associate con l'evento di uscita, quindi le azioni associate alla transizione possono aver luogo. Alla fine della relativa esecuzione si transita nel nuovo stato.

Si consideri per esempio il diagramma di fig. 10.7. La transizione verso lo stato denominato `Compimento` innesca simultaneamente le transizioni uscenti dai tre pseudostati iniziali presenti nelle altrettante regioni. La terminazione con successo dello stato si ottiene quando in tutte e tre le regioni si giunge allo stato finale. Ciò equivale a dire che lo studente è uscito indenne dalla parte pratica (costituita dal Laboratorio 1 e 2), ha realizzato una tesina e questa è stata accettata, e ha superato l'esame finale.

Figura 10.7 — *Sottostati concorrenti.*

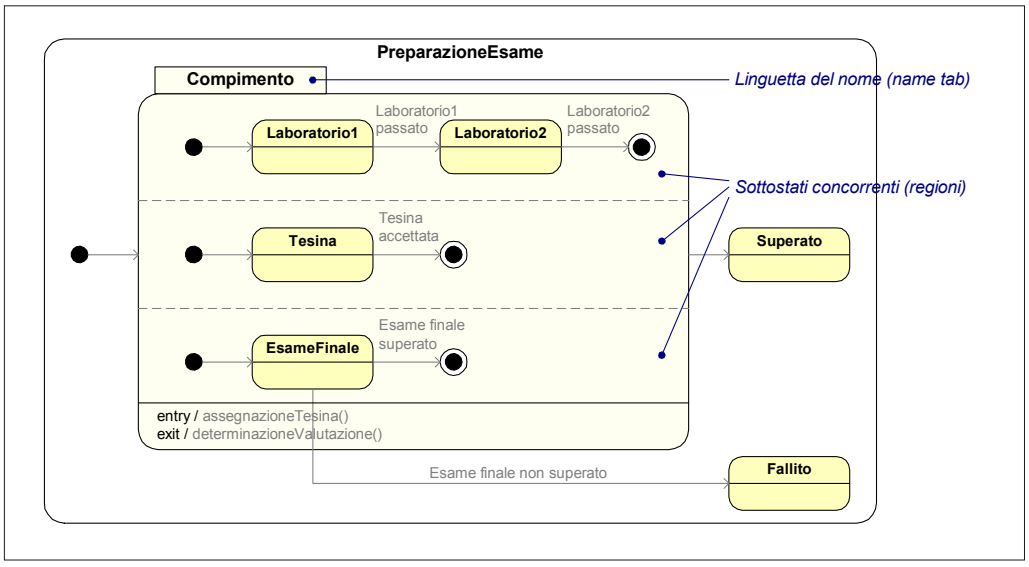
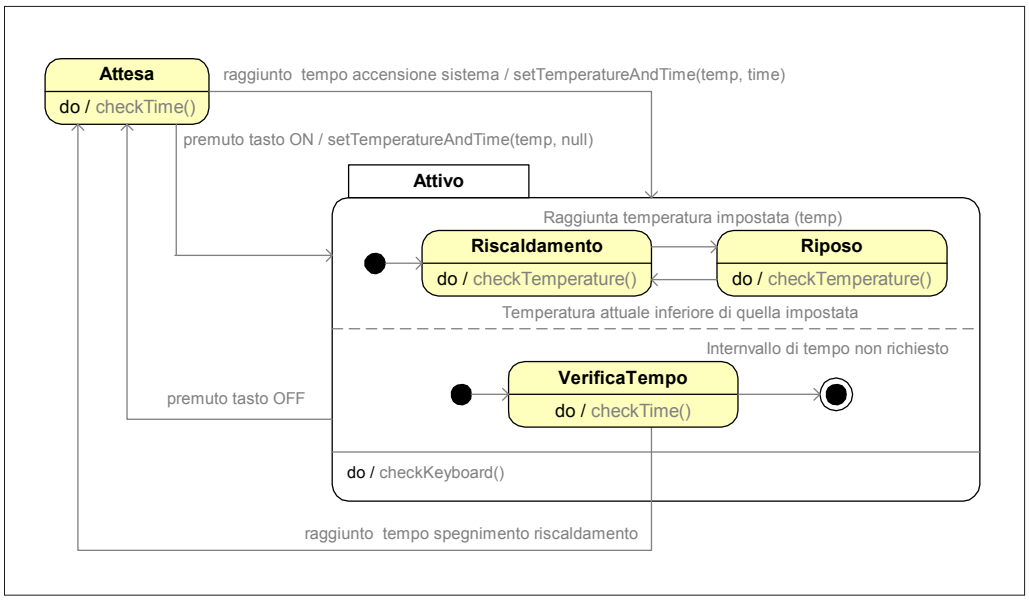


Figura 10.8 — *Ciclo di vita di un semplice termostato, il quale prevede due modalità di funzionamento: una temporizzata gestita da opportune programmazioni, e una attivata dall'utente ogniqualvolta voglia riscaldare a proprio piacimento gli ambienti.*



Nella fig. 10.8 è visualizzato il ciclo di vita di un semplice termostato. Qualora ci si trovi in un intervallo temporale selezionato per l'attivazione del riscaldamento, lo stato in esecuzione è quello composto denominato `Attivo`. Indipendentemente dallo stato di funzionamento dei riscaldamenti, qualora venga raggiunto l'orario di spegnimento degli stessi, lo stato `VerificaTempo` forza l'abbandono dello stato composto e quindi l'immediata terminazione dell'esecuzione delle varie attività.

Transizioni "stubbed"

Talune volte, per questioni di convenienza, è possibile rappresentare uno stato composto attraverso la notazione sintetica (fig. 10.9). Come si vedrà meglio nel paragrafo dedicato allo stile, questa tecnica offre tutta una serie di vantaggi: permette di gestire la complessità dei diagrammi, di rappresentare più agevolmente diagrammi complessi, ecc. La notazione sintetica prevede di rappresentare lo stato composto come un qualsiasi stato: nome, eventuali transizioni interne, ecc. ma, ovviamente, senza il riportare il dettaglio interno. Spesso si aggiunge l'icona dello stato composto (due anelli connessi per mezzo di un segmento) che, tipicamente, viene visualizzato nell'angolo in basso a destra. Si tratta di un accorgimento opzionale, ma che nella pratica si rivela molto utile.

Figura 10.9 — *Rappresentazione condensata di uno stato composto.*

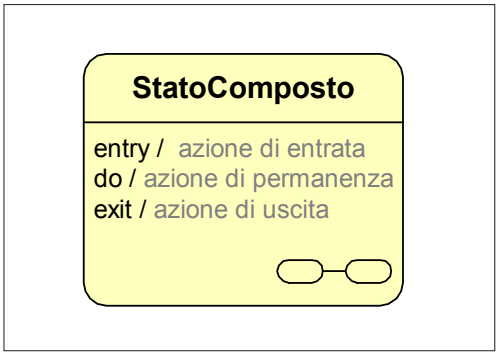
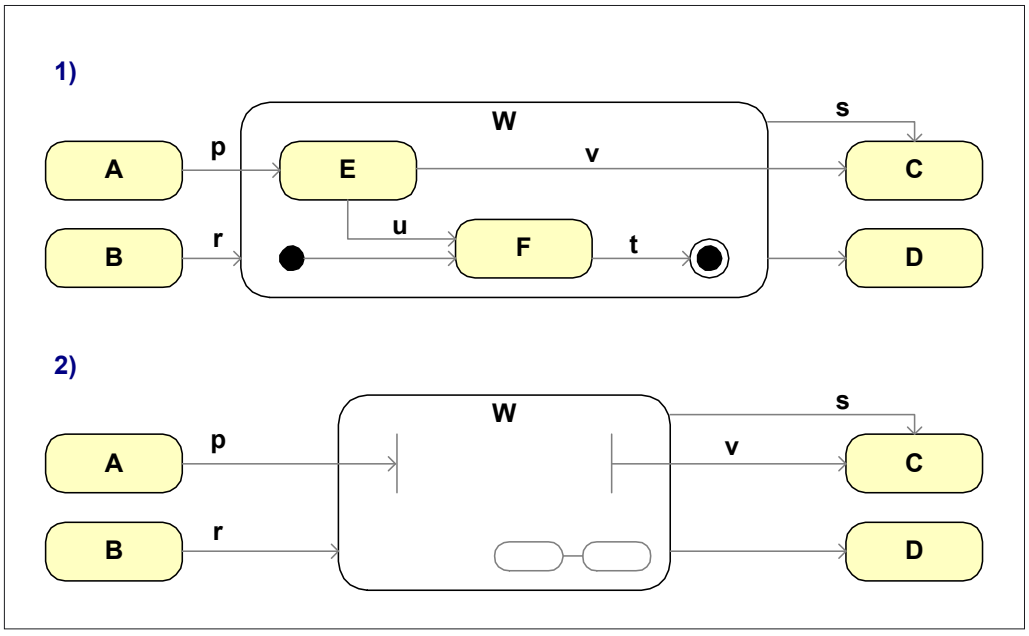


Figura 10.10 — *Utilizzo delle transizioni stubbed (tratto dalle specifiche ufficiali, figura3-79).*



La notazione sintetica però può generare diversi problemi di rappresentazione qualora vi siano transizioni destinate o a uno stato interno dello stato composto, o da questo generate. Qualora non si voglia mostrare alcun dettaglio, la transizione è collegata con lo stato più prossimo a quello non mostrato.

Queste transizioni possono (ma non è strettamente necessario) essere convenientemente mostrate come destinate o originate da appositi *stub* facenti funzioni degli stati non visualizzati. Questi stub sono rappresentati graficamente attraverso sottili segmenti orizzontali (barrette) visualizzati all'interno dell'elemento rappresentante lo stato composto (fig. 10.10). Tale accorgimento permette appunto di evidenziare che la transizione è diretta verso, o originata da, uno stato interno non visualizzato per questioni di rappresentazione grafica. Gli stub però non possono essere utilizzati al posto di pseudostati iniziali e stati finali.

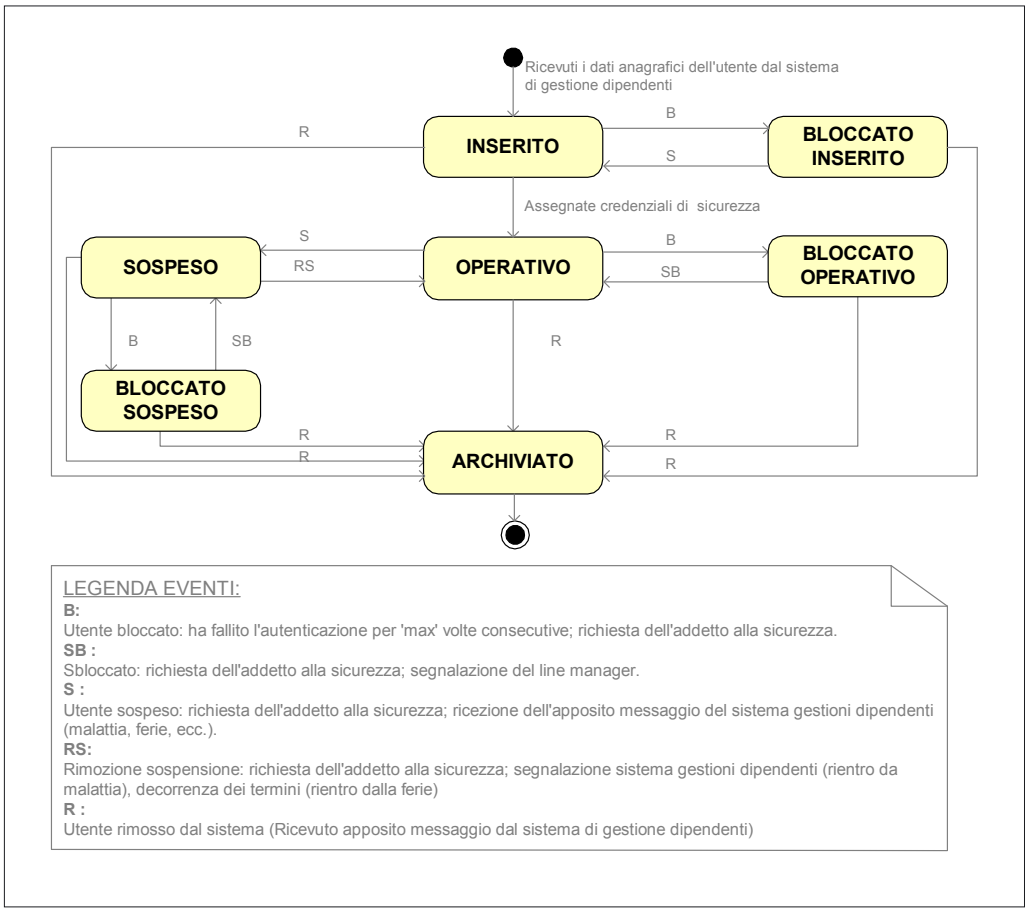
Stub visualizzati al posto di stati interni soppressi, terminazione di una transizione, sono legittimi e vantaggiosi. Lo stesso non si può dire per stub utilizzati come stati di origine di una transizione. Queste occorrenze dovrebbero essere ben ponderate perché una transizione è strettamente collegata con il corrispondente nodo di origine: pertanto, se questo non è visualizzato, è abbastanza legittimo attendersi che non lo siano neanche le transizioni partenti da esso.

Stato storico

In uno stato composto è possibile visualizzare un indicatore di stato storico (*history state indicator*), detto più semplicemente stato storico. Graficamente è mostrato attraverso la lettera H circondata da un cerchietto (fig. 10.12) e appartiene alla regione nella quale è visualizzato. Questo stato può avere diverse transizioni entranti, mentre deve prevederne una sola uscente, che, per definizione, non deve avere associata alcuna etichetta. Tale transizione termina nello stato a cui si passa per default (detto stato anteriore, *previous state*), ossia è lo stato in cui si transita, dallo stato storico, la prima volta in cui si entra nella corrispondente regione. Il comportamento a regime prevede che, una volta raggiunto lo stato storico, questo inneschi automaticamente l'unica transizione uscente, la quale termina nel sottostato in cui si trovava la macchina a stati presente nella regione di appartenenza dello stato storico, prima dell'ultima uscita dalla stessa. In altre parole, lo stato storico conserva memoria dell'ultimo stato visitato appartenente a un'apposita regione, prima dell'uscita dalla stessa.

In certi contesti è necessario disporre di uno stato storico profondo, in grado di riprendere l'esecuzione di un sottostato, a qualsiasi livello di annidamento, in cui si trovava la macchina a stati prima dell'uscita dallo stato composto. La differenza dallo stato storico semplice è che non si restringe l'attenzione ai soli stati dello stesso livello dell'indicatore dello stato storico. In questo caso la notazione grafica prevede un asterisco posposto alla `H` (`H*`). Per complicare le cose, una stessa regione può disporre di stato storico sia semplice che profondo.

Figura 10.11 — *Diagramma relativo al ciclo di vita di un utente di un sistema informatico.*



Si consideri il diagramma riportato in fig. 10.11. Dalla sua analisi, si evidenzia come sia stato necessario riportare una serie di stati fittizi (`BloccatoInserito`, `BloccatoOperativo` e `BloccatoSospeso`) per poter "ricordare" lo stato in cui ritornare, a seguito della rimozione di un blocco. Situazioni di questo tipo si risolvono ricorrendo all'utilizzo dello stato storico. Inoltre, è possibile evidenziare tutta una serie di eventi ripetuti. Al fine di evitare un diagramma a elevato grado di entropia, si è ricorso alla tecnica di evidenziare gli eventi attraverso un codice, riportandone la descrizione in un'apposita nota facente funzioni di legenda. Questa soluzione, se da un lato è utile per rendere il grafico meno caotico, dall'altro ne riduce la semplicità di fruizione e l'immediatezza.

Spesso, situazioni in cui vi siano tutta una serie di eventi ripetuti in diversi stati sono il segnale di un'organizzazione non ottimale del diagramma. Pertanto è opportuno rivedere lo stesso, eventualmente ricorrendo all'utilizzo di stati composti (fig. 10.12).

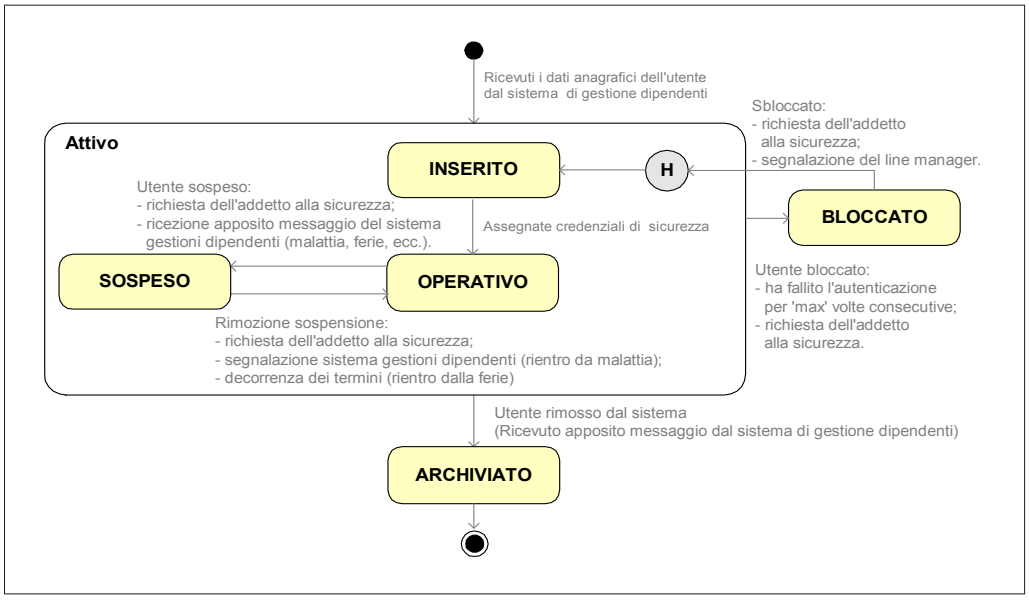
Gli stati storici sono rappresentati nel metamodello UML attraverso la metaclasse `PseudoState` il cui tipo è:

- `ShallowHistory` (storia superficiale): quando raggiunto, instrada la transizione verso lo stato appartenente allo stato composto che era attivo immediatamente prima che lo stesso stato composto fosse abbandonato. La ridirezione non avviene per eventuali sottostati dello stato attivo;
- `DeepHistory` (storia profonda): in questo caso la ridirezione può giungere fino a un qualsiasi livello di annidamento.

Eventi

Un evento è definito come "un avvenimento di una certa importanza". Nell'ambito dei diagrammi degli stati, si restringe l'attenzione ai soli eventi in grado di innescare una transizione di stato. Questi si possono presentare sotto diverse forme (fig. 10.13). In particolare si può trattare:

Figura 10.12 — *Diagramma relativo al ciclo di vita di un utente di un sistema informatico che sfrutta stati composti e stato storico.*



- di una condizione booleana, stabilita a tempo di disegno, che, assumendo il valore vero (`true`) dà luogo a un'istanza di tipo evento di cambiamento (`ChangeEvent` fig. 10.13). L'evento avviene esattamente nel momento in cui si ha il transito dal valore falso a quello vero. Spesso questi eventi vengono confusi con le condizioni di guardia: si tratta invece di concetti diversi. Una condizione di guardia non genera alcun evento, bensì è valutata solo dopo che questo è avvenuto. Il risultato della valutazione determina se l'evento possa o meno generare la corrispondente transizione. Nel caso di valore `false`, la transizione non avviene e l'evento viene perso.

- della ricezione di un segnale esplicito inviato da un oggetto a un altro. In questo caso si ha un'istanza della classe **evento segnale** (`SignalEvent`, fig. 10.13). Questo tipo di evento è evidenziato per mezzo della firma dell'evento stesso utilizzata come innesco di una transizione.

- della ricezione, da parte di un oggetto, di un'invocazione di un'operazione implementata come una transizione. In questo caso si ha un'istanza di tipo **evento di chiamata** (`CallEvent`, fig. 10.13).

- dello scadere di un definito intervallo di tempo, dopo la ricezione di un determinato evento, come per esempio l'entrata di uno stato concorrente, oppure il raggiungimento di una determinata data e orario. In questi casi si ha un'istanza di **evento temporizzato** (`TimeEvent`, fig. 10.13).

Figura 10.13 — *Frammento del metamodello UML relativo alla metaclasse* `Event`.

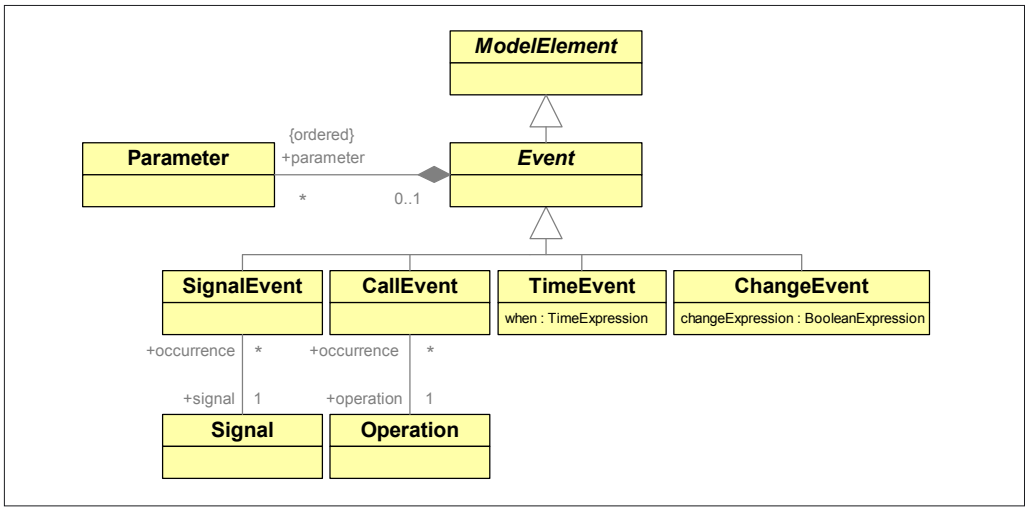
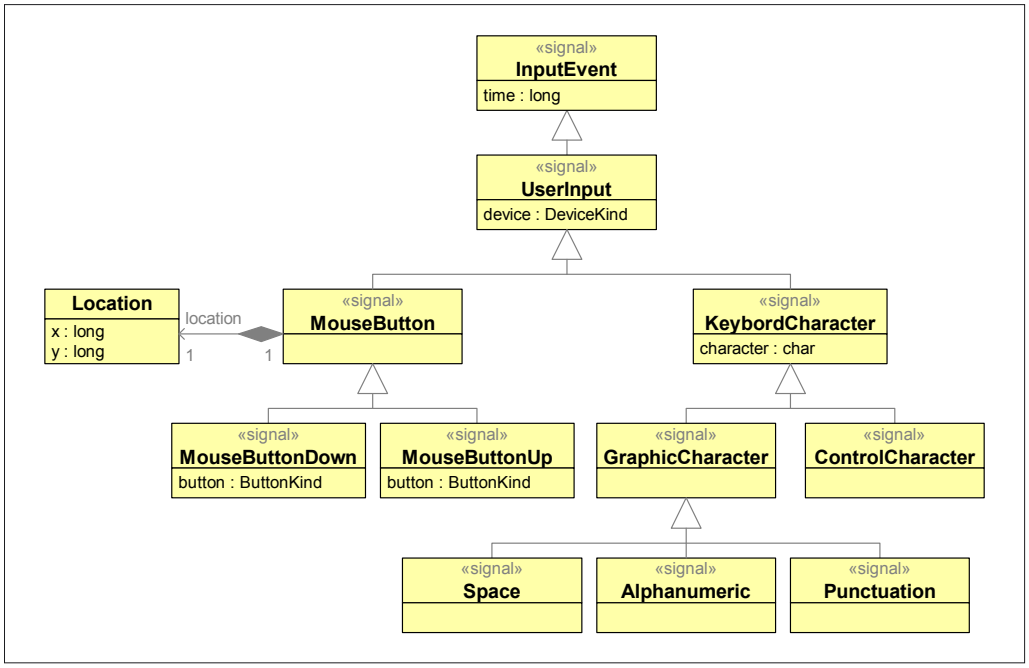


Figura 10.14 — *Esempio di classificazione di segnali utente (tratto dalle specifiche ufficiali, fig. 3-77). Diagrammi di questo tipo sono utilizzati per dichiarare i vari segnali.*



La dichiarazione di un evento ha validità (scope) all'interno del package dove è specificata, e può essere utilizzata in un diagramma degli stati per le classi che hanno visibilità all'interno dello stesso package. Un evento non può essere locale a una singola classe.

La notazione utilizzata per gli eventi segnale e di chiamata prevede una stringa del seguente formato:

```
event-name '(' comma-separated-parameter-list ') '
```

dove la stringa `comma-separated-parameter-list` ha il formato:

```
parameter-name : type-expression
```

Qualora sia necessario evidenziare eventi-segnale nei diagrammi delle classi, è possibile rappresentarli per mezzo dello stereotipo dell'elemento classe denominato `<<signal>>` (fig. 10.14). In questa rappresentazione, i parametri specificati nella stringa `comma-separated-parameter-list` diventano attributi della relativa classe. Si consideri per esempio il diagramma di fig. 10.14 e in particolare la classe `MouseButtonDown`. La relativa visualizzazione testuale, in un diagramma degli stati, potrebbe assumere una for-

ma del tipo: `MouseButtonDown (time:long, device:DeviceKind, location:Location, button:ButtonKind)`.

I segnali si prestano a essere inseriti in opportune gerarchie di classificazione (un segnale sottoclasse di un altro, ecc.). Ciò ha un significato molto importante e preciso: un'occorrenza di un "sottoevento" innesca eventuali transizioni dipendenti da eventi antenati di quello verificatosi. Considerando ancora l'esempio riportato poco sopra, un'occorrenza dell'evento "pressione del pulsante del mouse" (`MouseButtonDown`), ha la potenzialità di innescare eventuali transizioni associate con gli eventi: `MouseButtom`, `UserInput` e `InputEvent`.

Un evento temporizzato si presta a essere specificato per mezzo della parola chiave `after` seguita dall'espressione da valutare, a tempo di modellazione, indicante l'intervallo di tempo specificato. Alcuni esempi sono: `after(300 secondi)` oppure `after(300 secondi dall'ultima interazione con l'utente)`. Qualora non sia specificato esplicitamente il momento di inizio dell'avvio del "cronometro", questo viene assunto coincidente all'entrata nello stato. Gli altri tipi di eventi temporizzati possono essere dichiarati attraverso la parola chiave `when`. Per esempio, `when(date = 01/01/2000)`.

Transizioni

Transizioni semplici

Una transizione semplice è una relazione tra due stati indicante che una istanza nel primo stato entrerà nel secondo, eseguendo determinate azioni, al verificarsi di un prestabilito evento, qualora le eventuali condizioni specificate (condizioni di guardia) siano verificate. Quando avviene il cambiamento di stato, si dice che la transizione è "scatenata" (nella letteratura americana si usa il termine *fired*).

L'innesco della transizione è fornito da un opportuno evento riportato nell'etichetta della transizione stessa. Gli eventi possono dichiarare una serie di parametri accessibili sia dalle azioni specificate nelle transizioni, sia da quelle corrispondenti agli eventi di entrata e uscita specificati, rispettivamente, nello stato destinazione e in quello di partenza (sorgente). Gli eventi sono processati uno alla volta e vengono ignorati qualora non diano luogo a alcuna transizione. La situazione opposta, cioè l'innesco di più transizioni da parte di uno stesso evento nel contesto di una stessa regione sequenziale, prevede che solo una sia scatenata. Qualora non esistano criteri per assegnare la priorità, la selezione della transizione da scatenare avviene randomicamente.

La notazione utilizzata per le transizioni è, chiaramente, la stessa riportata nella sezione delle transizioni interne, pertanto assume il formato:

```
event-name '(' comma-separated-parameter-list ')' '[' guard-condition ']' '/' action-expression
```

La condizione di guardia (`guard-condition`) è un'espressione booleana che, tipicamente, utilizza i parametri specificati nell'evento scatenante la relativa transizione e i col-

legamenti dell'oggetto di cui la macchina a stati descrive il ciclo di vita. Le condizioni di guardia possono anche specificare esplicite verifiche degli stati concorrenti appartenenti a determinati stati composti, oppure stati di un qualche oggetto raggiungibile (per esempio: `in state1`, oppure `not in state2`). I nomi degli stati possono essere riportati in maniera completa, indicando tutti gli stati che lo includono, separati da una doppia ripetizione del carattere "due punti" (per esempio, `State1::State2::State3`). Questa notazione è particolarmente utile qualora uno stesso nome sia utilizzato per diversi stati appartenenti a diverse regioni concorrenti della stessa macchina a stati.

L'espressione azione (`action-expression`) è eseguita quando la transizione viene scatenata e l'eventuale condizione di guardia risulta soddisfatta. Questa espressione può prevedere una serie di operazioni, attributi, collegamenti dell'oggetto a cui la macchina a stati si riferisce, parametri relativi all'evento innescante oppure qualsiasi altra caratteristica accessibile. L'espressione azione deve essere eseguita completamente prima di poter considerare qualsiasi altra azione. Questa può comprendere una sequenza di azioni, includendo anche quelle in grado di generare eventi come, per esempio, inviare segnale o invocare operazioni. La selezione del linguaggio da utilizzare per dichiarare queste azioni è completamente demandato al modellatore.

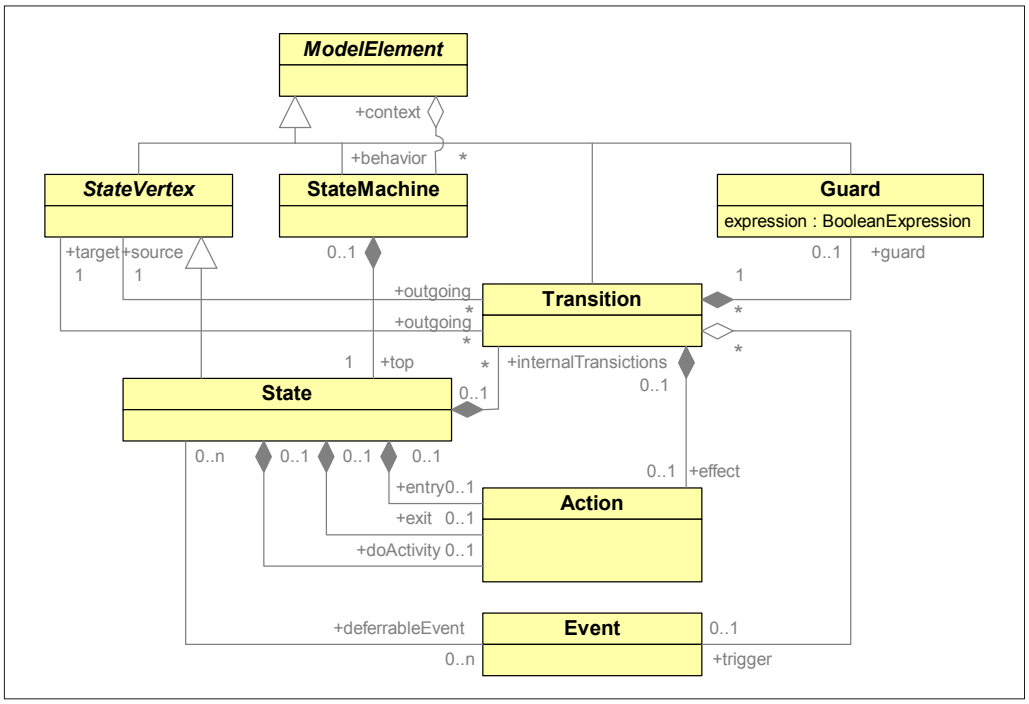
Si consideri l'evento `MouseButtonDown` dichiarato nel diagramma di fig. 10.13. Una transizione innescata da questo potrebbe assumere la forma:

```
MouseButtonDown(time, device, location, button)
   [buttom == ButtonKind.Left && location in window] /
   object := window.selectObject(location);
   object.highlight()
```

Gli eventi devono appartenere a una delle categorie specificate nella fig. 10.12. Nella maggior parte dei casi è possibile discriminare il tipo dell'evento dalla sintassi con cui è specificato. A questa regola fanno eccezione gli eventi segnale e di chiamata. In questo caso la differenziazione deve avvenire con mezzi alternativi, quali per esempio la dichiarazione in un apposito diagramma delle classi oppure l'utilizzo di note.

Il diagramma di fig. 10.15, riproducente la porzione di metamodello relativa all'elemento `Transition`, è stato presentato in quanto permette di mostrare sinteticamente e formalmente — la potenza dei modelli — concetti particolarmente interessanti presentati nel corso dei paragrafi precedenti. In primo luogo si può notare che sia le transizioni interne, sia quelle tra stati sono in effetti la stessa cosa: sono modellate dal medesimo elemento (`Transition`). La differenza risiede nel fatto che nel caso di transizioni interne l'elemento `Transition` è parte di un solo stato (recita il ruolo di `internal Transition` nella composizione con la metaclasse `State`) e la relativa esecuzione non genera l'uscita e il rientro nello stato stesso (da non confondersi quindi con le autotransizioni). Una transizione specifica gli stati attraverso i quali avviene la transizione

Figura 10.15 — *Porzione di metamodello relativa all'elemento* `Transition`.



(sorgente e destinazione, nel metamodello `source` e `target`), può disporre di una sola condizione di guardia (visualizzata attraverso la composizione con l'elemento `Guard`), può prevedere un evento innescante (metaclasse `Event` nel ruolo di `trigger`) e può specificare un'azione opzionale da eseguire qualora innescata (metaclasse `Action` nel ruolo `effect`). Per finire si può notare che le transizioni interne degli stati sono mostrate attraverso opportune relazioni di composizione con la metaclasse `Action`, in particolare sono predefiniti gli eventi: `entry`, `exit` e `doAction` che, qualora associate a opportune azioni, sono eseguite, rispettivamente, all'atto dell'entrata, dell'uscita e durante la permanenza nello stato .

Transizioni concorrenti

Una transizione è definita concorrente qualora dia luogo a una sincronizzazione di due o più transizioni originate da diversi stati concorrenti, e/o, caso opposto, generi una suddivisione del flusso di controllo in più flussi concorrenti destinati a due o più stati concorrenti. Pertanto una transizione concorrente può essere originata e terminare in diversi stati concorrenti.

La semantica della transizione concorrente stabilisce che quando tutti gli stati concorrenti sorgenti sono raggiunti, la transizione concorrente può aver luogo e, una volta scatenata, vengono raggiunti tutti gli stati destinazione.

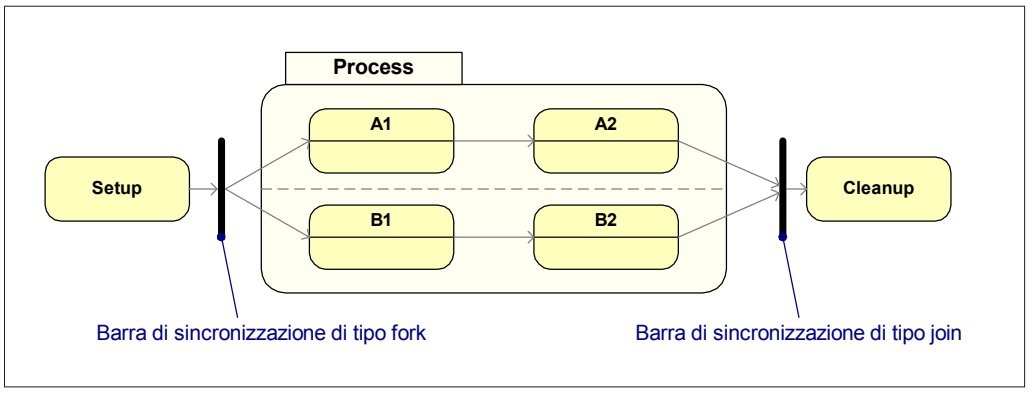
La notazione grafica prevede l'utilizzo di un segmento visualizzato con tratto spesso, denominato barra di sincronizzazione (*synchronization bar*). Questa può essere utilizzata per rappresentare sincronizzazioni, suddivisioni del flusso di controllo o entrambe (fig. 10.16). Qualora si decida di ricorrere alla barra di sincronizzazione, la stringa di transizione deve essere mostrata nei pressi della barra stessa e non sulle singole frecce entranti o uscenti.

> Nel metamodello UML la barra di sincronizzazione è un'istanza dell'elemento `PseudoState` (fig. 10.5) il cui tipo (`kind`) può essere:
>
> - `fork`, qualora la transizione in arrivo sia suddivisa in diverse transizioni concorrenti in uscita (avviene la suddivisione del flusso del controllo);
> - `join`, situazione opposta: avviene la sincronizzazione del flusso del controllo, ossia più transizioni concorrenti si riuniscono in una sola.
>
> La condizione in cui avvenga una sincronizzazione con immediata suddivisione del flusso di controllo è visualizzata per mezzo di due barre di sincronizzazione unite attraverso un singola transizione.

Transizioni composte

Secondo la definizione ufficiale dello UML, una transizione collega esattamente due "stati" (esattamente due `StateVertex`, fig. 10.5). Poiché però questi stati possono essere istanze dell'elemento `PseudoState`, transienti per definizione (stati di passaggio), è

Figura 10.16 — *Esempio di utilizzo della barra di sincronizzazione. (Diagramma tratto del testo delle specifiche ufficiali 3-78).*



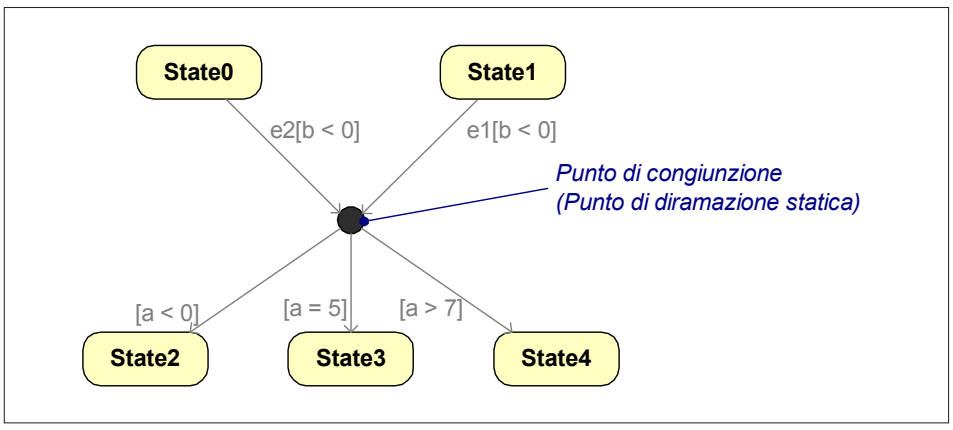
possibile rappresentare catene di transizioni che possono essere eseguite nel contesto di un singolo passo di esecuzione. Questo tipo di transizioni sono dette transizioni composte (*compound transition*).

Nel modellare i diagrammi di stato, spesso può rivelarsi utile condividere dei segmenti di transizioni composte, per esempio, due o più distinte transizioni possono unirsi e continuare per un percorso comune, condividendo azioni ed eventualmente terminando nello stesso stato di destinazione. In altre situazioni invece, può rivelarsi utile suddividere una transizione in diversi cammini mutuamente esclusivi.

Questi esempi si prestano a essere rappresentati graficamente in maniera "fattorizzata" in cui alcune parti del percorso sono condivise. Questa tecnica si dimostra molto utile per modellare comportamenti dinamici adattativi. Un esempio classico è fornito da un singolo evento che può terminare in un insieme di stati differenti, e la selezione dipende dal risultato dell'esecuzione di un'azione eseguita dopo l'innesco di una transizione composta.

La suddivisione e il ricongiungimento dei percorsi che si ottengono attraverso le transizioni composte, sono un concetto diverso da suddivisione e ricongiungimento di transizioni concorrenti che si ottengono con le barre di sincronizzazione (illustrate nel paragrafo precedente). In questo caso, sia gli stati sorgente sia quelli destinazione non sono stati concorrenti e si ha un concetto di selezione di un singolo percorso tra quelli in alternativa (*multiplexing*). Pertanto non si originano o riuniscono più flussi concorrenti.

Figura 10.17 — *Diagramma utilizzato per rappresentare un punto di congiunzione della tipologia diramazione statica (tratto dalle specifiche ufficiali, fig. 3-81).*



Le transizioni composte si prestano a essere rappresentate graficamente mediante i **punti di congiunzione** (*junction point*). Questi permettono a varie transizioni provenienti da diversi stati (o pseudostati) non concorrenti di ricongiungersi oppure, a una specifica transizione proveniente da un singolo stato non concorrente di diramarsi in diversi percorsi alternativi. In questo modo è possibile far condividere alle transizioni entranti/uscenti i percorsi originati/destinati al punto di congiunzione. Per quanto concerne la rappresentazione grafica, nei diagrammi degli stati, i punti di congiunzione sono rappresentati per mezzo di piccole circonferenze nere (fig. 10.17). Una notazione alternativa consiste nell'utilizzare i rombi decisionali, che invece sono preferiti nel contesto dei diagrammi di attività.

Figura 10.18 — *Ecco la macchina a stati relativa allo stato composto denominato* `Lavorazione` *di fig. 10.6. Si noti come l'utilizzo del punto di diramazione statica abbia permesso di conferire maggiore enfasi alla decisione da prendersi a seguito dell'attività di revisione.*

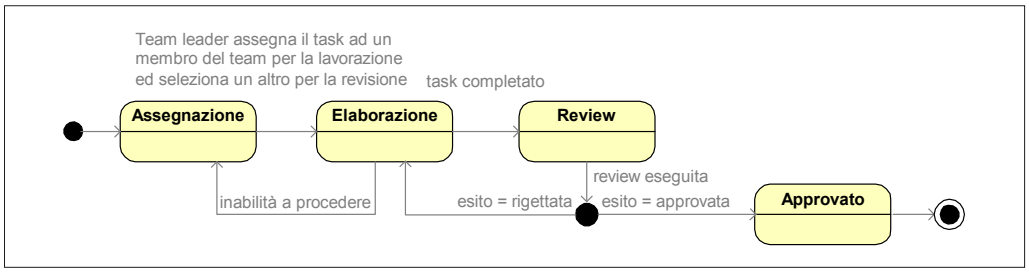
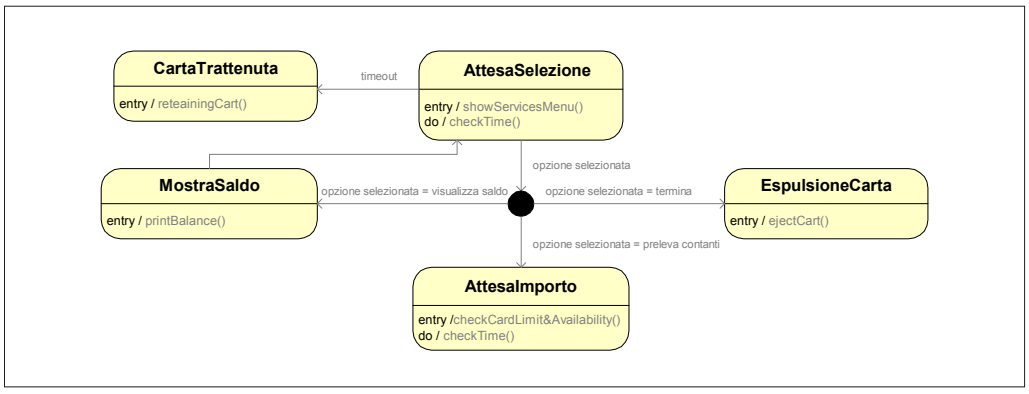


Figura 10.19 — *Frammento rivisitato della macchina a stati inerente il funzionamento di un bancomat vista in fig. 10.4. Questa parte è relativa alla selezione di un'opzione da parte dell'utente. L'utilizzo del punto di diramazione ha evidenziato le diverse alternative.*



I punti di congiunzione si dividono in due categorie: **punti di diramazione statica** (*static branch point*) e **punti di selezione dinamica** (*dynamic choice point*).

I primi sono caratterizzati dal fatto che le transizioni uscenti dal punto di congiunzione posseggono delle condizioni di guardia, tipicamente a mutua esclusione, il cui esito può essere stabilito fin dal momento in cui si innesca la transizione che porta al punto di congiunzione. Questa tecnica permette di evitare fitti grafi di connessioni tra stati: la notazione tradizionale prevede di riportare tante transizioni quanti sono i segmenti entranti nel punto di congiunzione, moltiplicate per il numero di quelle uscenti. In ogni transizione è poi necessario riportare condizioni di guardia formate dal concatenamento, per mezzo dell'operatore logico *and*, di tutte le condizioni presenti nelle transizioni attraversate dal particolare percorso sostituito. Per esempio, considerando il diagramma di fig. 10.17, la notazione classica avrebbe richiesto sei transizioni. Per quanto riguarda lo `State0` sarebbe stato necessario dar luogo alle seguenti transizioni, dirette, rispettivamente, allo:

`State2`, con condizione di guardia: `e2[b < 0 and a < 0];`

`State3`, con condizione di guardia: `e2[b < 0 and a = 5];`

`State4`, con condizione di guardia: `e2[b < 0 and a = 7];`

Per quanto riguarda lo `Stato1` la situazione sarebbe stata pressoché analoga.

Questa tipologia di punti di congiunzione è definita diramazione statica poiché la decisione di quale percorso intraprendere è stabilita prima che una transizione sia innescata. In altre parole, le condizioni presenti nelle condizioni di guardia possono essere valutate da subito, e non contengono parametri scaturiti dall'esecuzione di azioni presenti su particolari percorsi di opportune transizioni.

Da quanto detto è facile intuire la differenza con i punti di selezione dinamica, i quali permettono di modellare decisioni dinamiche: solo una volta arrivati nel punto di congiunzione, o meglio, dopo aver innescato la transizione diretta a tale punto, è possibile valutare le condizioni di guardia delle transizioni uscenti e quindi, solo allora, è possibile selezionare la transizione di uscita. Questo equivale a dire che, nelle condizioni di guardia delle transizioni uscenti dal punto di congiuntura, sono presenti criteri il cui dominio contiene valori ottenuti dall'esecuzione delle azioni corrispondenti alle transizioni entranti nello pseudostato.

Per esempio, nel diagramma di fig. 10.20, la selezione di quale transizione innescare una volta giunti al punto di selezione dinamica, dipende dal valore *a* determinato dall'esecuzione dell'azione presente nella transizione entrante nello pseudostato. In questo caso quindi, non è possibile selezionare il percorso a priori.

I punti di selezione dinamica sono rappresentati graficamente attraverso una circonferenza vuota.

Figura 10.20 — *Diagramma utilizzato per rappresentare un punto di congiunzione della tipologia selezione dinamica (tratto dalle specifiche ufficiali, fig. 3-82).*

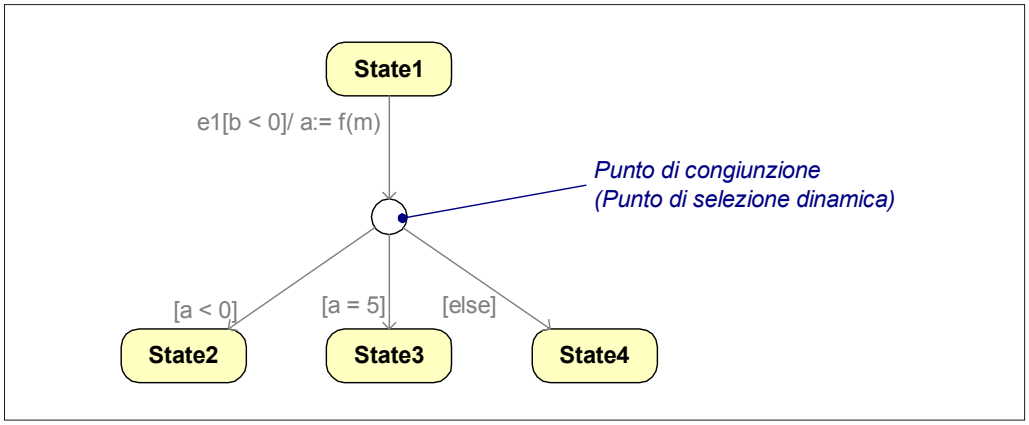
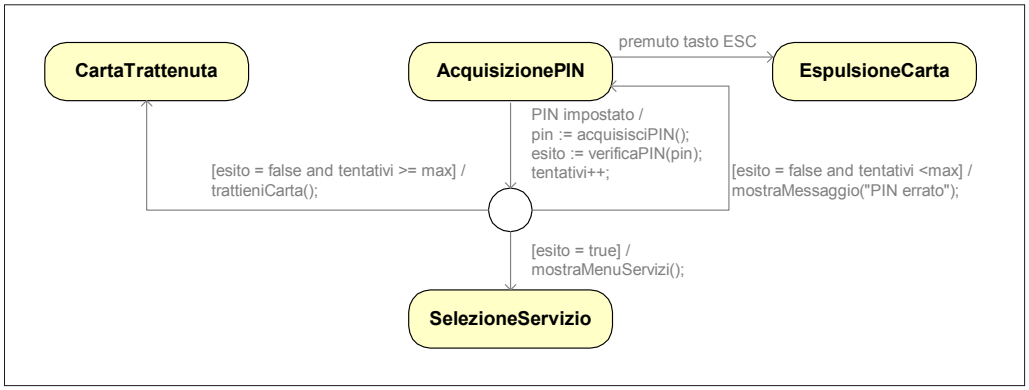


Figura 10.21 — *Il diagramma rappresenta la porzione della macchina a stati inerente il funzionamento di un bancomat, relativo alla validazione del PIN. In questo caso si è scelta una tecnica alternativa, rispetto a quella mostrata nel diagramma di fig. 10.4.*



Stati di sincronizzazione.

Uno stato di sincronizzazione (*Synch State*) è utilizzato per sincronizzare macchine a stati (concorrenti) di diverse regioni di uno stato composto concorrente. Il suo utilizzo avviene in congiunzione con le barre di sincronizzazione. Questo accorgimento è necessario per assicurarsi che la macchina a stati di una particolare regione lasci uno specifico stato, prima che la macchina a stati di un'altra regione entri in un determinato stato.

L'innesco della transizione di uscita dello stato di sincronizzazione può essere limitato, specificando esplicitamente il valore massimo della differenza tra il numero delle transizioni che lasciano lo stato di sincronizzazione e quelle che vi entrano. In sostanza si può dichiarare un limite al parallelismo ammesso.

Gli stati di sincronizzazione sono visualizzati graficamente attraverso una circonferenza con all'interno un numero positivo specificante il limite suddetto. Nel caso in cui non si voglia imporre alcun limite, si utilizza il carattere asterisco (fig. 10.22).

Stato di "sottomacchina"

Uno stato di sottomacchina (*submachine state*) rappresenta l'invocazione di una macchina a stati definita da qualche altra parte. Si tratta di un concetto molto simile a quello delle invocazioni di macro: costituisce una rappresentazione grafica che implica una specificazione complessa all'interno di un'altra specificazione.

In generale, è possibile accedere a un qualsiasi sottostato di una macchina a stati invocata, quantunque l'accesso preferenziale sia quello relativo allo pseudostato iniziale. Analogamente, è possibile uscire da qualsiasi sottostato di una macchina invocata, sebbene il caso tipico sia quello della stessa macchina che raggiunge il proprio stato finale. Entrate e uscite non standard vengono visualizzate per mezzo di speciali stati stub.

Dal punto di vista della rappresentazione grafica, lo stato di sottomacchina è rappresentato utilizzando la notazione classica, con in più, nel compartimento dedicato alle transizioni interne, la dichiarazione di inclusione (`include`). Questa parola chiave permette di dichiarare la sottomacchina a stati invocata.

Figura 10.22 — *Stato di sincronizzazione.*

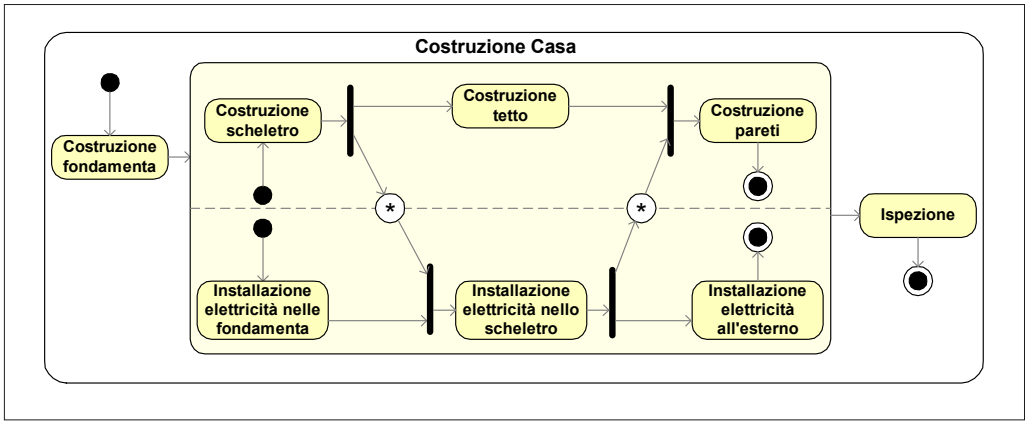
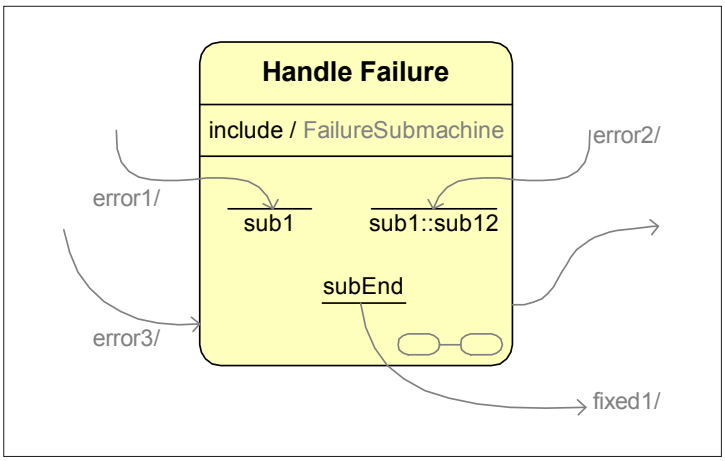


Figura 10.23 — *Diagramma rappresentante uno stato di sottomacchina dedicato alla gestione delle anomalie (tratto dalle specifiche ufficiali, fig. 3-83). A tal fine utilizza una sottomacchina a stati,* `FailureSubmachine`*, definita in un'altra parte.*



A questo punto è opportuno porre attenzione ai due concetti: lo stato di sottomacchina incapsula l'invocazione a una macchina a stati finiti, definita sottomacchina.

Qualora sia necessario accedere a stati particolari della macchina a stati invocata, o processare transizioni di uscita da predefiniti stati non finali, è possibile ricorrere agli stati di stub. In questo contesto, è necessario assegnare a ogni stub il nome del relativo stato della macchina invocata. Qualora, poi, il sottostato sia annidato, occorre premettere al nome dello stato il relativo percorso: elenco degli stati in cui transitare per giungere a quello di interesse, separati dalla ripetizione del carattere due punti (`::`).

Chiaramente, qualora l'accesso e l'uscita della macchina a stati invocata avvenga attraverso le vie standard (pseudostato iniziale, stato finale), non è necessario ricorrere a nessuna rappresentazione particolare.

Diversi stati di sottomacchina, invocanti una stessa sottomacchina a stati, possono essere inclusi svariate volte nel contesto di una stessa macchina a stati con, ovviamente, diverse configurazioni di entrata e di uscita e differenti azioni associate agli eventi di entrata e uscita.

Dall'analisi del diagramma di fig. 10.23 è possibile notare la presenza della parola chiave `include` che dichiara la macchina a stati invocata (`FailureSubmachine`). Nel contesto di uno stesso diagramma è possibile inserire diversi stati di sottomacchina invocante la medesima sottomacchina con diverse configurazioni.

Lo stato di sottomacchina `Handle Failure` è in grado di processare diversi eventi. In particolare, riceve le transizioni innescate dagli eventi:

- `error1`, gestito dallo stato `sub1` appartenente alla sottomacchina invocata. Il nome dello stato riportato nello stub non dispone di percorso, indicando che tale stato non è annidato;

- `error2`, fornito allo stato `sub1::sub12`. In questo caso il nome dello stato è annidato. Quindi per giungere allo stato `sub12` è necessario transitare per quello `sub1`;

- `error3`: questa transizione giunge direttamente allo stato di sottomacchina (`FailureSubmachine`) e quindi genera l'avvio standard della macchina a stati.

Infine, si può notare che il sottostato `subEnd` della sottomacchina a stati innesca la transizione `Fixed1/`.

Utilizzo

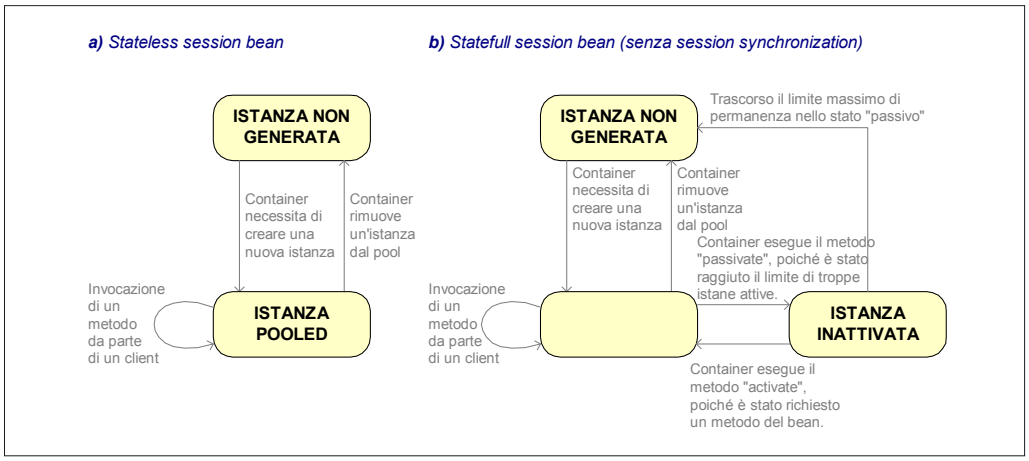
I diagrammi delle macchine a stati sono una notazione dello UML particolarmente utile e potente atta a rappresentare particolari aspetti dinamici del sistema. Come si è visto, questo formalismo affonda le proprie radici nella teoria delle macchine a stati. Tale eredità ne consente l'utilizzo in svariate branche dell'ingegneria: in tutti quei casi in cui la realtà si presta a essere modellata attraverso oggetti il cui ciclo di vita evolve attraverso stati ben definiti.

La notazione dei diagrammi degli stati si configura anche come un formidabile strumento per fini didattici: senza conoscerne le regole formali, li si utilizza fin dalle scuole medie. Si consideri per esempio i diagrammi di cambiamento degli stati utilizzati in fisica. In ambienti più informatici, li si utilizza per illustrare diversi concetti, come per esempio la gestione delle transizioni, il ciclo di vita dei componenti (fig. 10.24), ecc.

Nel contesto dei processi formali di sviluppo del software sono utilizzabili praticamente in tutti gli stadi. Durante la fase di analisi dei requisiti, è conveniente utilizzare la notazione dei diagrammi di stato per definire in maniera semplice, formale e intuitiva, il ciclo di vita di oggetti business particolarmente importanti. Per esempio, nei diagrammi di figg. 10.11 e 10.12, il formalismo dei diagrammi è stato utilizzato per definire il ciclo di vita degli utenti di un sistema informatico. Si tratta di un utilizzo molto utile per definire sinteticamente e formalmente requisiti funzionali particolarmente importanti. In un sistema bancario, per esempio, si potrebbero utilizzare per specificare il ciclo di vita di particolari contratti (trade); in un sistema di commercio elettronico, si potrebbe realizzare una macchina a stati finiti rappresentante l'evoluzione di un ordine, ecc.

Tipicamente gli utenti, a meno di ricorrere a particolari notazioni o a un eccessivo livello di dettaglio, non sono impauriti da questa notazione, che, invece, percepiscono come familiare.

Figura 10.24 — *Utilizzo della macchina a stati per visualizzare il ciclo di vita dei session bean. Da notare che il ciclo di vita dei message driven bean è del tutto equivalente a quello dei session bean stateless, con la differenza che le invocazioni dell'unico metodo (*`onMessage`*) avvengono da parte del container ogni qualvolta riceve un messaggio per il quale il message drivem bean si sia precedentemente registrato.*



Se si decide di utilizzare la notazione delle macchine a stati nelle fasi iniziali del processo, è necessario ricordare che tra i fruitori di maggiore importanza dei modelli di questa fase ci sono gli utenti del sistema, i quali, tipicamente, hanno scarse conoscenze di concetti tecnici e OO. Quindi, è necessario realizzare diagrammi con un grado di astrazione abbastanza elevato, evitando il più possibile particolari artifici permessi dalla notazione e concetti a elevato grado tecnico. In breve, è necessario tendere a forme che ne agevolino la fruizione da parte di un pubblico non tecnico. Per le ottimizzazioni c'è sempre tempo: le versioni iniziali saranno dettagliate durante le successive fasi del processo di sviluppo.

La notazione dei diagrammi degli stati si presta anche a descrivere altre entità a maggiore carattere tecnico. Per esempio diversi tecnici consigliano di utilizzarli per descrivere la navigazione dell'interfaccia utente. L'autore del presente testo non trova particolarmente vantaggioso l'impiego della notazione della macchina a stati per questi fini, per i quali la notazione dei diagrammi di collaborazione si dimostra decisamente più congeniale. Un utilizzo particolarmente consigliato consiste, invece, nel rappresentare il ciclo di vita di particolari meccanismi presenti nel modello di disegno, come particolari "engine" (motori), sistemi di gestione dell'allocazione delle risorse, parser, ecc. In questo contesto i dia-

grammi sono destinati a un pubblico con elevate conoscenze tecniche e OO (architetti, disegnatori e programmatori), pertanto vengono meno tutte le limitazioni e gli accorgimenti imposti nell'utilizzo della notazione durante le frasi precedenti.

Stile

Nomi significativi

La prima linea guida inerente lo stile non poteva non essere riferita ai nomi degli stati. In particolare, si consiglia di utilizzare nomi semplici, brevi e descrittivi. Nella maggior parte dei casi, la formula migliore consiste nel ricorrere a verbi riportati al presente, sebbene, anche verbi al participio passato risultano spesso convenienti. Nel caso della macchina a stati relativa alla gestione dei *Non Conformity Requirement* (fig. 10.6), si è preferito utilizzare nomi di stato con verbi al participio passato: `Inserito`, `Analizzato`, `Completato`, ecc.

Per ciò che riguarda i nomi delle transizioni, questi dovrebbero essere costituiti da verbi che ricordino l'evento che le ha generate. Verbi coniugati al passato sono più rispondenti al fatto che l'evento che ha innescato la transizione è già scaturito.

I nomi degli stati, nonché il linguaggio utilizzato per specificare le varie azioni, devono essere consistenti con la fase in cui sono utilizzati i relativi diagrammi. Per esempio, durante l'analisi dei requisiti è consigliabile utilizzare, quanto più possibile, nomi consistenti con le regole del business e evitare di specificare azioni attraverso un linguaggio di programmazione, mentre in modelli a carattere più strettamente tecnico è preferibile utilizzare un linguaggio consistente con tale fase; quindi azioni specificate attraverso il linguaggio di programmazione selezionato (o in una pseudo versione), stati con nomi legati all'implementazione tecnica, ecc. Per esempio nella macchina a stati relativa agli stateless session bean si è utilizzato il termine *pooled*.

Posizionare adeguatamente i nomi

Nei diagrammi degli stati sono presenti diversi nomi, principalmente quelli degli stati e delle transizioni. È opportuno conferire ai primi particolare enfasi, magari ricorrendo a un font più grande di quello utilizzato per gli altri elementi.

Per quanto attiene ai nomi delle transizioni, qualora possibile, è opportuno specificarli vicino allo stato di origine.

Evitare stati anonimi

Nel realizzare diagrammi degli stati, è importante evitare gli stati anonimi, quantunque sia permesso dalla notazione. Molto spesso la presenza di questi stati confonde le idee e tende a nascondere importanti regole di business. Se non si riesce proprio a pensare a un nome adatto, può essere il campanello di allarme che c'è qualche problema nel modello: più stati condensati in uno solo, un elemento che non ha senso come stato a sé stante, ecc.

Qualora invece si ritenga assolutamente necessario rappresentare stati anonimi, è fortemente consigliato, nel contesto di uno stesso diagramma, limitarne il numero all'unità. Sebbene, secondo lo standard UML, si tratti di stati diversi, nella mente dei lettori tende a ingenerarsi l'idea che si tratti dello stesso stato.

Sindacare l'assenza di pseudostati iniziali e degli stati finali

Tutte le macchine a stati di primo livello, non incorporate in altre macchine a stati, devono necessariamente prevedere un pseudostato iniziale e uno stato finale. La mancanza della visualizzazione di tali stati non è sempre un errore; in ogni modo, qualora omessi, è consigliabile rivedere il modello.

Spesso capita di analizzare modelli con vari stati finali, dotati di diversi nomi. Non è infrequente il caso in cui si ricorra a tale strategia in quanto si è omesso di rappresentare esplicitamente importanti stati (quelli il cui nome è stato assegnato allo stato finale).

Curare la posizione degli pseudostati iniziali e degli stati finali

Molti autori consigliano di sistemare gli pseudostati iniziali e gli stati finali in modo tale da favorire una lettura dei diagrammi da sinistra verso destra: si posiziona lo pseudostato iniziale in alto a sinistra e lo stato finale in basso a destra. Come al solito, si tratta di un'impostazione confacente alle abitudini di lettura della cultura occidentale. Esempi di questa impostazione si possono trovare in vari diagrammi (figg. 10.6, 10.7, ecc.).

Quantunque si tratti di un ottimo consiglio, anche un posizionamento a fruizione verticale — sarà questa la famosa visione giapponese? — con lo pseudostato iniziale posto in cima e quello finale posto in basso, risulta molto naturale (figg. 10.3, 10.11, 10.12).

Controllare i "buchi neri" e gli "stati dei miracoli"

Con i termini di "buchi neri" e "stati dei miracoli", Scott Ambler, si riferisce, rispettivamente, a stati destinatari di transizioni che non generano transizioni uscenti e, caso opposto, stati da cui escono transizioni ma nei quali non ne giunge alcuna.

È importante controllare bene stati di questo tipo perché, tipicamente, sono frutto della dimenticanza di qualche transizione o di una modellazione non ottimale.

Chiaramente a questa regola fanno eccezione gli pseudostati iniziali e gli stati finali.

Utilizzare stati e transizioni composte

Nel caso in cui un diagramma degli stati risulti particolarmente complesso, è consigliato eseguire un'attività di revisione volta a verificare se sia possibile ridurne la complessità. Un meccanismo utilizzabile a tal fine consiste nel ricorrere a stati e/o transizioni composte. In particolare è consigliabile considerare l'introduzione di stati composti in diverse circostanze. Un esempio è quando diversi stati prevedono transizioni, magari innescate da uno stesso evento, dirette verso medesimi stati destinazione; un altro è il caso contrario, cioè quando diversi stato sono destinatari di transizioni relative a medesimi eventi. Un

esempio dell'introduzione di uno stato composto, atto a ridurre la complessità del sistema è quello relativo al ciclo di vita degli utenti di un sistema informatico (figg. 10.11 e 10.12).

L'introduzione di transizioni composte va considerata in tutti quei casi con una fitta rete di transizioni, o quando pochi stati siano origine e/o destinazione di molte transizioni.

Da tenere presente però, che l'introduzione di queste notazioni non sempre è priva di effetti collaterali. Spesso, specialmente le transizioni composte sono di difficile comprensione per un pubblico di non tecnici. Pertanto, nel caso di diagrammi degli stati utilizzati in fase di analisi dei requisiti, è opportuno ponderarne l'utilizzo. Un altro vantaggio nel ricorrere a transizioni composte è che si evidenziano condizioni particolarmente degne di nota (figg. 10.18 e 10.19)

Utilizzare stati storici

Non è infrequente il caso in cui in diversi diagrammi degli stati si inseriscano svariati stati fittizi, inclusi solo per necessità di modellazione. Talune volte risulta una situazione obbligatoria, mentre in altri casi si tratta esclusivamente di un errato utilizzo delle transizioni (specie di quelle interne), oppure del mancato utilizzo di uno stato storico. Come visto nell'esempio del diagramma di fig. 10.11, relativo al ciclo di vita degli utenti di un sistema informatico, l'introduzione dello stato storico ha permesso di semplificare la rappresentazione del diagramma senza renderlo particolarmente ostico (fig. 10.12).

Come nel caso precedente, si tratta di una notazione non sempre alla portata di un pubblico non esperto, pertanto è consigliato ponderarne l'utilizzo specie nelle primissime fasi del processo di sviluppo del software.

Utilizzare rappresentazioni collassate

Molto spesso il diagramma relativo a una macchina a stati risulta decisamente complesso anche dopo aver utilizzato stati e transizioni composte. In questi casi, probabilmente è opportuno verificare la possibilità di visualizzare determinati stati composti senza illustrarne il dettaglio interno, demandandolo ad appositi diagrammi. Molto spesso questa tecnica permette di semplificare il diagramma, agevolandone la fruizione. In sostanza, si modella la macchina secondo la più classica delle fruizioni top-down: un diagramma mostra gli stati composti di primo livello; eseguendo un doppio click su uno di essi, se ne vede il dettaglio interno, e così via. Un'eccezione è data dal caso in cui per visualizzare sinteticamente stati composti è necessario ricorrere a svariati stub. In questi casi, bisogna ben ponderare i vantaggi offerti dalla rappresentazione top-down della macchina a stati, con la necessità di mostrare molti elementi grafici aggiuntivi e non sempre di facile comprensione.

Porre attenzione all'utilizzo delle transizioni interne

Qualora in un determinato stato si evidenzino transizioni interne, bisogna porre bene attenzione che queste si applichino effettivamente a tutti i casi. Alcune volte può capitare

di analizzare modelli aventi stati con azioni associate a transizioni di entrata e di uscita, per poi rendersi conto che le azioni devono essere eseguite solo in casi particolari. Spesso è possibile risolvere il problema con opportune condizioni di guardia, mentre altre volte è opportuno rivedere le azioni associate alle transizioni che terminano in tale stato (per le azioni agli eventi di entrata) e che ne escono (azioni associate agli eventi di uscita) al fine di verificare la possibilità di aggiungervi le azioni che accadono solo in particolari casi di entrata e/o uscita dallo stato.

Evitare di rappresentare macchine a stati randomiche

Qualora, nell'ambito di uno stato (o di una regione sequenziale nel caso di stati compositi), uno stesso evento possa innescare diverse transizioni, la notazione delle macchine a stati prevede che solo una sia scatenata. La selezione avviene con criterio casuale.

Situazioni del genere, benché valide dal punto di vista della notazione, tipicamente, generano macchine a stati con comportamento non prevedibile. Pertanto, a meno di situazioni molti particolari, come per esempio situazioni business basate su eventi aleatori, è opportuno evitare di rappresentare macchine a stati con transizioni sfruttanti criteri aleatori.

Per gli stessi motivi, è importante evitare di rappresentare punti di giunzione con condizioni di guardia "sovrapposte". Si supponga di avere una condizione basata sul credito di un cliente, e due transizioni relative all'importo, sarebbe non opportuno dichiarate condizioni di guardia del tipo: `credito <= 0`, `credito >= 0`.

Utilizzare punti di diramazione statica

Spesso capita di dover rappresentare uno stato con diverse transizioni uscenti, relative allo stesso evento, che si differenziano per la condizione di guardia.

Si consideri l'esempio di una persona che tenta di prelevare un certo importo da un bancomat. In questo caso potrebbe accadere che l'importo selezionato ecceda il limite previsto dalla carta. Ciò potrebbe essere modellato, visualizzando lo stato `seleziona importo` e due transizioni uscenti dallo stesso stato:

- importo selezionato con condizione di guardia `credito <= limite`;
- importo selezionato con condizione di guardia `credito > limite`;

In situazioni del genere, è possibile evidenziare una sola transizione di uscita con un punto di diramazione statica, nelle cui transizioni di uscita riportare la condizione di guardia.

L'utilizzo di questa tecnica, tipicamente, permette di conferire maggiore risalto alla condizione della diramazione e tende a semplificare il livello di complessità dei diagrammi prodotti.

I diagrammi di attività

Introduzione

La notazione dei diagrammi di attività rappresenta una variante dei diagrammi degli stati, in cui le proprietà dell'elemento stato sono ridotte.

> Nel metamodello l'elemento che rappresenta i diagrammi di attività (`ActivityGraph`) è una specializzazione della metaclasse che identifica i diagrammi delle carte di stato (`StateMachine`). Dall'analisi del metamodello (fig. 10.25) e dei relativi vincoli è possibile evidenziare come non si tratti di una "vera e propria" specializzazione, bensì di una restrizione della semantica di un insieme di elementi, al fine di renderne più agevole l'utilizzo nel contesto dei diagrammi di attività. In altre parole si tratta della definizione di notazioni più semplici e convenienti da utilizzare nel contesto dei diagrammi di attività.

Come visto in precedenza, uno stato rappresenta una condizione durante il ciclo di vita di un'entità, nel corso del quale questa soddisfa determinate condizioni, esegue specifiche operazioni e attende l'occorrenza di opportuni eventi. Nel contesto dei diagrammi delle attività, la quasi totalità degli stati utilizzati rappresenta l'esecuzione di attività e, pertanto, la maggior parte delle transizioni è innescata implicitamente dall'evento di completamento delle predette attività, mentre le condizioni soddisfatte durante la permanenza nello stato hanno a che fare con l'esecuzione delle attività stesse.

Per esempio nel diagramma di fig. 10.26 è presente un insieme di particolari stati, il cui unico compito consiste nell'eseguire il comando visualizzato internamente. Una volta terminata l'esecuzione, viene innescata automaticamente la transizione di uscita.

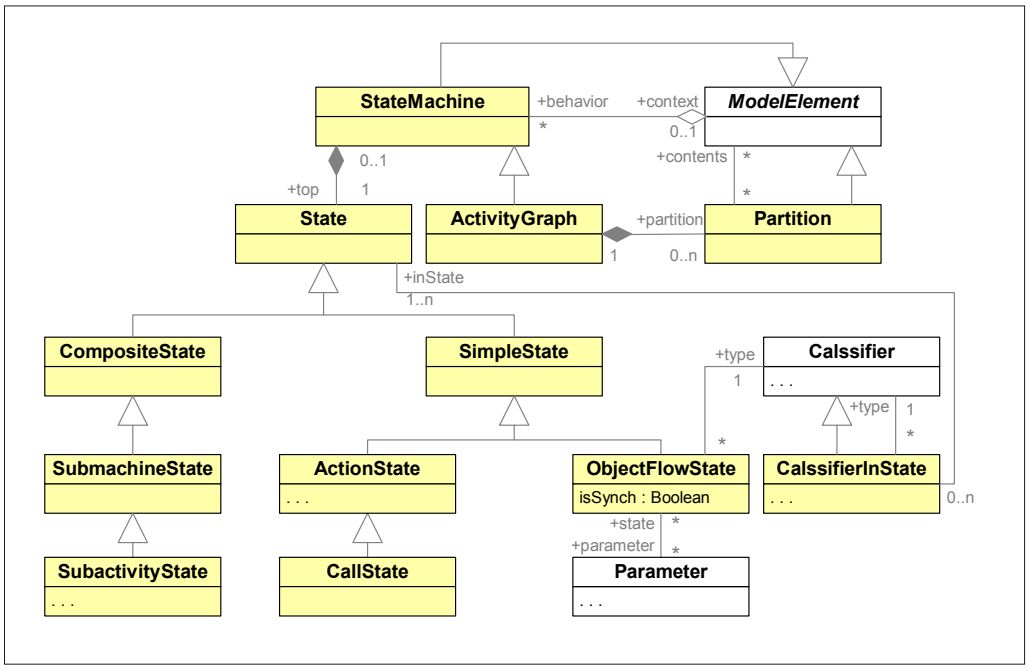
I diagrammi delle attività, analogamente a quelli di stato, sono associati a un classificatore, come un caso d'uso, un package, l'implementazione di un'operazione, ecc. e permettono di modellarne gli aspetti dinamici. In questo caso l'attenzione è focalizzata sui processi computazionali, dei quali si enfatizza il controllo del flusso e l'evoluzione degli oggetti durante l'esecuzione delle azioni. Pertanto, la quasi totalità degli eventi rappresenta il completamento delle azioni eseguite internamente. In qualche modo quindi, i diagrammi delle attività risultano antitetici a quelli degli stati che focalizzano l'attenzione sulle transizioni generate da eventi esterni e quindi sono preferiti per la modellazione di sistemi in cui gli eventi asincroni recitano un ruolo importante.

Dal punto della notazione, un diagramma di attività rappresenta un grafo che connette particolari specializzazioni di stati attraverso, peculiari transizioni.

Metamodello dei diagrammi di attivita

Nella fig. 10.25 è riportato il frammento del metamodello UML relativo ai diagrammi di attività. Sebbene possa sembrare piuttosto complicato, i vari ele-

Figura 10.25 — *Frammento del metamodello UML relativo ai diagrammi di attività.*



menti sono illustrati nel corso della presentazione del capitolo e tutto assumerà una luce diversa… Come al solito, la mancata comprensione del presente diagramma non costituisce un limite alla comprensione della notazione dei diagrammi di attività.

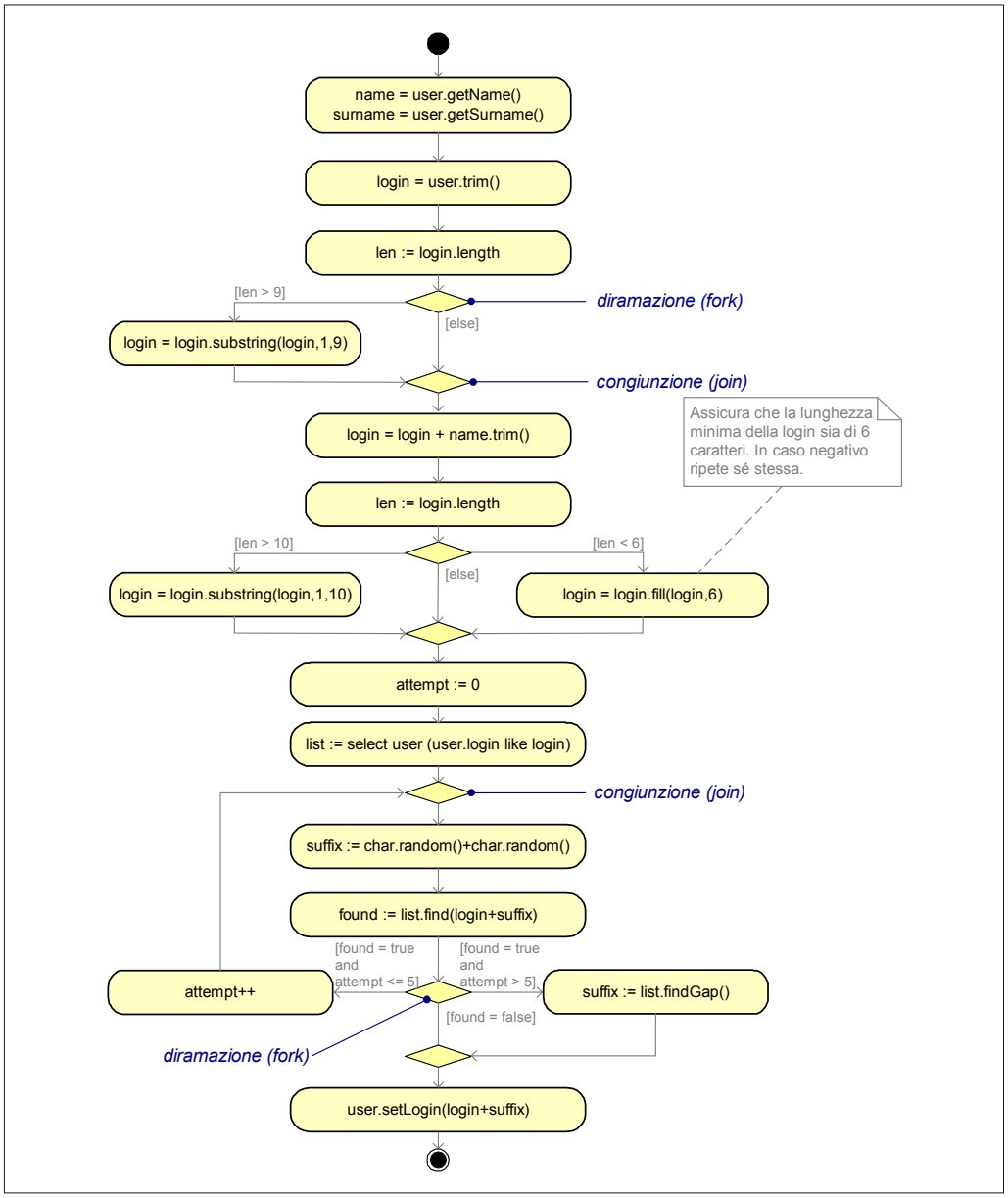
Il diagramma mostra in maniera evidente come il formalismo dei diagrammi delle attività sia a tutti gli effetti una specializzazione di quello dei diagrammi di stato, e anche gli elementi che vi compaiono sono particolari specializzazioni degli stati utilizzati nei diagrammi di stato. Un elemento che potrebbe destare qualche perplessità è quello denominato `Partition` (un activity diagram può essere costituito da più partizioni). Come si vedrà successivamente, si tratta dell'elemento che permette di rappresentare le *swimlane* ("corsie di nuoto").

Elementi dei diagrammi di attività

Stato di azione (ActionState)

Uno stato di azione è una specializzazione di uno stato semplice (fig. 10.25). Si tratta di una sua versione semplificata, caratterizzata dal disporre di un'azione associata all'evento di entrata e con, almeno, una transizione di uscita innescata (implicitamente) dalla termi-

Figura 10.26 — *Utilizzo dei diagrammi di attività per modellare algoritmi di calcolo. In particolare, l'algoritmo rappresentato in figura serve per generare stringhe di login univoche da assegnare ai nuovi utenti del sistema. Queste stringhe sono ottenute dalla concatenazione di una parte del cognome, più una porzione del nome con in aggiunta due caratteri random.*



nazione della suddetta attività. Tipicamente a questi stati non viene assegnato alcun nome e pertanto rappresentano istanze di stati anonimi.

Gli action state sono pertanto caratterizzati dall'esecuzione di una o più azioni di entrata, il cui completamento dell'esecuzione innesca almeno una transizione di uscita. Questa può disporre di azioni, di condizioni di guardia (e quindi possono essere diverse) mentre non è ammesso includerne la firma.

In breve, uno stato di azione rappresenta l'esecuzione di un'attività atomica, per esempio l'invocazione di un'operazione e, pertanto, si presta a modellare un passo di un algoritmo e/o un'attività di un particolare processo (fig. 10.26).

Altre proprietà degli stati di azione consistono nel non disporre né di azioni corrispondenti all'evento di permanenza nello stato (`doActivity`), né di eventi di uscita, né di transizioni innescate da eventi espliciti. Queste restrizioni non si desumono dal diagramma delle classi riportato in fig. 10.25, bensì dai vincoli aggiuntivi. Qualora sia necessario utilizzare una di queste proprietà, è opportuno ricorrere ad altri tipi di stati.

Graficamente, gli stati azione sono mostrati con una notazione molto simile a quella degli stati tradizionali: rettangolo dagli angoli smussati, con all'interno visualizzata l'espressione di azione (`action-expression`). La notazione utilizzabile per specificare tale espressione è totalmente demandata al modellatore: l'unico vincolo è che deve essere univoca all'interno dello stesso diagramma. Come al solito, qualora si realizzi un diagramma di attività durante la fase di analisi dei requisiti, è opportuno utilizzare un linguaggio semplice, vicino al linguaggio tecnico degli utenti, mentre, qualora li si utilizzi per documentare un algoritmo di calcolo, o un processo del sistema appartenente al modello di disegno, è opportuno utilizzare il linguaggio di programmazione selezionato o, meglio, un opportuno pseudocodice. Per esempio, poiché il diagramma di fig. 10.26 modella un algoritmo, il formalismo selezionato per rappresentare le azioni da compiere è un semplice pseudocodice. Qualsiasi sia la notazione selezionata, questi passi possono utilizzare le proprietà (attributi e relazioni) dell'oggetto di cui si modellano le dinamiche. Sempre con riferimento al diagramma di fig. 10.26, molte azioni utilizzano parametri appartenenti all'operazione descritta, come `user`, `login`, `attempt`, ecc.

Sebbene questo stato sia disegnato per i diagrammi di attività, può essere chiaramente utilizzato anche nel contesto dei diagrammi di stato (si tratta di una versione dello stato semplice a semantica ristretta).

Stato di chiamata (CallState)

Lo stato di chiamata rappresenta un ulteriore discendente dello stato semplice, passando per lo stato di azione. In particolare, gli stati di chiamata sono caratterizzati dal disporre di un'unica azione, associata all'evento di entrata nello stato, che si risolve in un'invocazione.

Consistentemente con quanto illustrato fin'ora, la relativa semantica comporta un'ulteriore restrizione, introdotta al fine di disporre di un altro elemento atto a modellare circostanze ben definite. Come si vedrà successivamente, questo elemento è stato introdotto per

Figura 10.27 — *Esempi di stati di chiamata corredati con le operazioni invocate.*

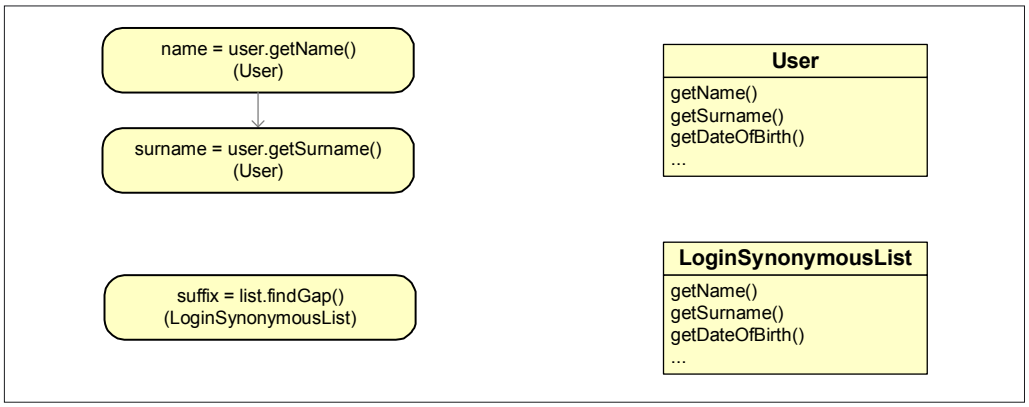
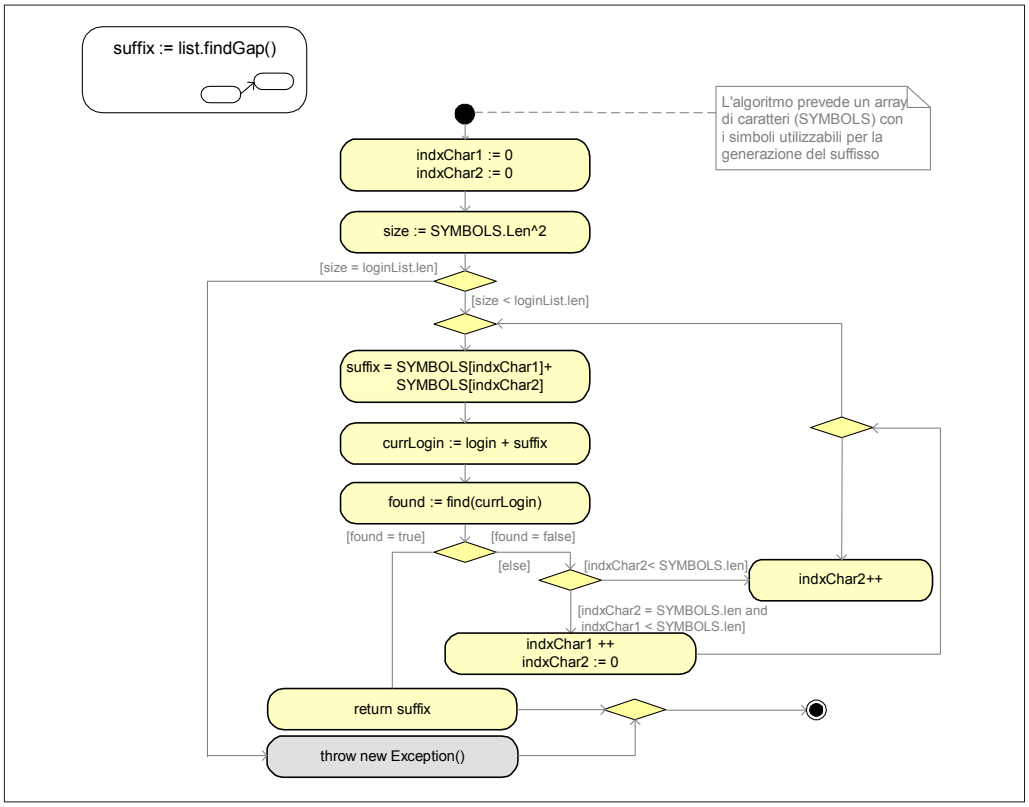


Figura 10.28 — *Rappresentazione di uno stato di sottoattività con il diagramma di dettaglio.*



rimuovere ambiguità nella notazione utilizzata durante la modellazione dei flussi degli oggetti, con particolare riferimento alle azioni che acquisiscono degli input da quelli che li producono.

La notazione grafica è del tutto equivalente a quella degli stati di attività con una sola variante: nel rettangolo smussato è riportato sia il nome dell'operazione, sia quello del classificatore che la implementa, racchiuso da parentesi tonde.

Per esempio, nel diagramma di attività relativo al calcolo della stringa di login (fig. 10.26) si sarebbero potuti evidenziare gli stati di chiamata riportati in fig. 10.27.

Stato di sottoattività

Analogamente a quanto visto per gli stati di sottomacchina, la notazione dei diagrammi di attività prevede gli stati di sottoattività.

> Nel metamodello UML la metaclasse `SubactivityState` eredita da quella `SubmachineState` (fig. 10.25).

Questo stato è utilizzato per invocare l'esecuzione di grafi di attività definiti in opportuni diagrammi specificati in altre parti del modello. Pertanto rappresenta l'esecuzione di un insieme di passi non atomici che possono avere una certa durata: non è infrequente il caso di avere dei processi di sincronizzazione.

Il comportamento degli stati di sottoattività non presenta particolari sorprese: quando vi si accede, il controllo passa al diagramma di attività annidato, il quale viene eseguito come da prassi (dallo pseudostato iniziale, fino allo stato finale che devono essere necessariamente inclusi). Da questo stato poi si esce quando si raggiunge lo stato finale del diagramma di attività invocato, oppure quando viene innescata una transizione di uscita dello stato di sottoattività. Da tener presente che quest'ultima alternativa non dovrebbe essere frequente giacché gli stati dei diagrammi di attività non dovrebbero possedere transizioni innescate da eventi espliciti.

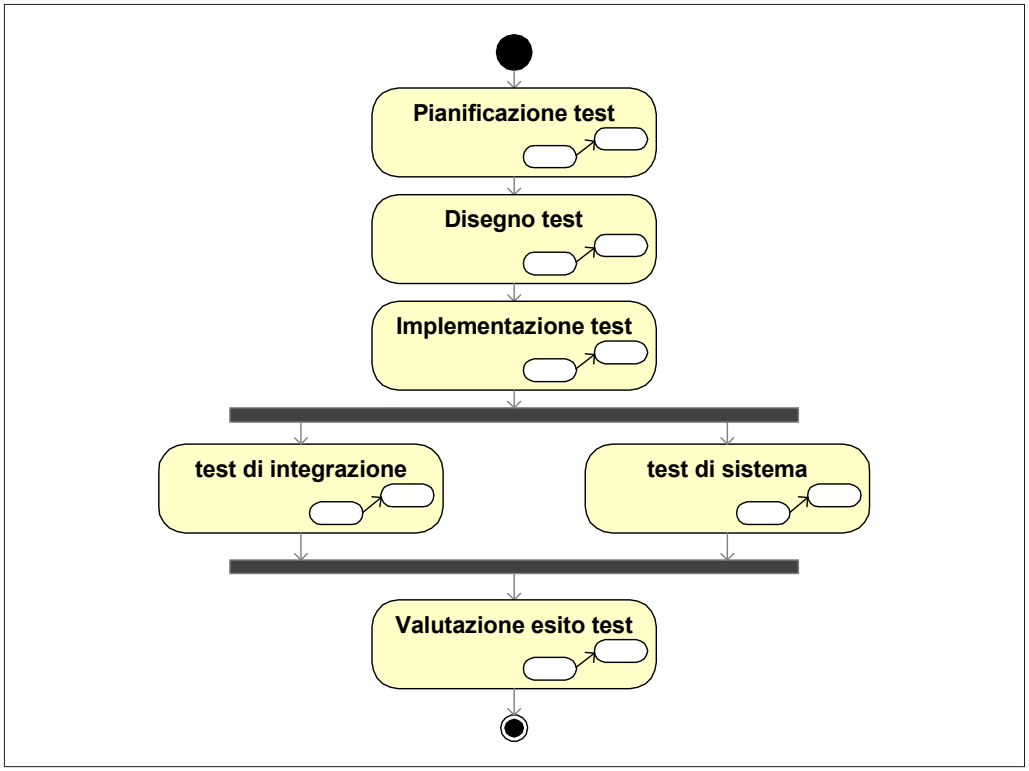
Da un punto di vista pratico, questo stato può essere immaginato come un'invocazione di macro: ogni istanza di uno stato di sottoattività può essere sostituita dall'intero diagramma degli stati invocato. Uno stesso diagramma di attività può essere incluso diverse volte in un altro, attraverso opportuni stati di sottoattività.

La notazione grafica degli stati di sottoattività prevede il medesimo rettangolo con gli angoli smussati con l'aggiunta di un'apposita icona (due stati connessi attraverso una freccia) visualizzata nell'angolo in basso a destra. In questo caso è prassi associare un nome agli stati di sottoattività visualizzati all'interno del grafico.

Per esempio, nel diagramma di attività relativo al calcolo della stringa di login (fig. 10.26), lo stato azione relativo all'individuazione di un suffisso disponibile si prestava a essere rappresentato per mezzo di un apposito stato di sottoattività, con il diagramma specificante le azioni riportate in un'altra sede, come evidenziato nell'immagine di fig. 10.28.

Nell'immagine di fig. 10.29 è rappresentato un esempio di un diagramma di attività utilizzato per rappresentare una versione del workflow inerente la fase di test (ebbene i sistemi devono essere testati…). Come si può notare, tutti gli stati visualizzati sono istanze di stati di sottoattività e, pertanto, è lecito attendersi la definizione dettagliata in un apposito diagramma di attività non visualizzato in questo testo. L'utilizzo di questa tecnica permette di organizzare gerarchicamente i diagrammi di attività secondo un classico approccio top-down: si forniscono le informazioni generali, effettuando un doppio click in uno stato di sottoattività se ne visualizza il diagramma di dettaglio, eventualmente si effettua un'ulteriore doppio click in un altro stato di sottoattività e via discorrendo.

Figura 10.29 — *Rappresentazione del workflow di test attraverso la notazione dei diagrammi di attività. Nei processi iterativi e incrementali, queste attività devono essere eseguite al completamento di ogni iterazione. Chiaramente non tutte richiedono di eseguire il processo di disegno e di implementazione di test.*



Elementi di decisione e di ricongiunzione

Nei diagrammi di attività, analogamente a quanto illustrato per i diagrammi di stato, le decisioni sono mostrate attraverso apposite condizioni di guardia associate alle transizioni. Qualora si realizzino percorsi composti, per mezzo di opportuni segmenti connessi da appositi pseudostati, le condizioni di guardia permettono di selezionare i segmenti che compongono un determinato percorso tra quelli in alternativa; in altre parole, gli pseudostati e le condizioni di guardia permettono di suddividere il flusso del controllo.

In questo caso si fa riferimento al concetto della diramazione e non a quello dell'esecuzione concorrente. Nel contesto dei diagrammi di attività, gli elementi decisionali sono visualizzati attraverso la notazione classica del rombo (fig. 10.26).

Nel caso dei diagrammi di attività bisogna ricordarsi che gli elementi decisionali non sono in grado di compiere computazioni (eseguire azioni), contrariamente a quanto avveniva per i diagrammi di flusso. In questo contesto, rappresentano un conveniente punto di transizione e diramazione del flusso. Pertanto l'azione da cui scaturisce il valore (o i valori) utilizzati per attuare la diramazione deve essere specificata in un apposito stato antecedente l'elemento decisionale e a esso connesso attraverso un'opportuna transizione.

In ogni transizione uscente dallo pseudostato di decisione, viene visualizzata l'apposita condizione di guardia che ne abilita la percorrenza. Il formalismo dei diagrammi di attività prevede la condizione di guardia predefinita, denominata `else`, utilizzabile per al più una transizione di uscita da uno stesso pseudostato. Il significato è quello tradizionale, qualora l'esito della valutazione delle restanti condizioni restituisca un valore `false`, allora viene abilitata la transizione etichettata con la stringa `else`.

Lo stesso simbolo (il rombo) può essere utilizzato per riunire flussi precedentemente divisi. In questo caso il simbolo prende il nome di *merge* ed è caratterizzato dall'avere due o più transizioni entranti e una sola uscente (fig. 10.26).

Questo elemento, sia nella versione di suddivisone del flusso, sia in quella di ricongiunzione, è completamente consistente con il punto di diramazione presentato nel contesto dei diagrammi degli stati, pertanto nel metamodello UML è rappresentato dall'elemento `Pseudostate` il cui tipo è `junction`.

Swimlane ("corsie di nuoto")

Un diagramma di attività può essere suddiviso in un insieme di partizioni, ognuna delle quali è costituita da una collezione di elementi non condivisi.

> Nel metamodello UML le *swimlane* sono rappresentate dalla metaclasse `Partition` (fig. 10.25). In particolare, la possibilità di organizzare un diagramma di attività tramite partizioni è fornita dalla composizione tra la classe `ActivityGraph` e quella `Partition`, mentre gli elementi appartenenti a una partizione sono mostrati per mezzo della relazione `contents`.

Queste partizioni sono denominate *swimlane* (= "corsie di nuoto") e permettono di organizzare i diagrammi delle attività secondo precisi criteri. Tipicamente, le swimlane sono associate a determinate unità organizzative (come il sistema, sue parti, attori, particolari ruoli, ecc.) e pertanto permettono di enfatizzare l'allocazione delle attività o, se si preferisce, l'assegnazione delle responsabilità, alle diverse unità.

Graficamente sono visualizzate attraverso rettangoli (con riportati all'interno gli stati appartenenti) dotati di un nome, visualizzato in alto (fig. 10.30).

Le swimlane sono visualizzate consecutivamente (il primo lato di quella successiva è sovrapposto al secondo di quella precedente) secondo un ordine deciso dal modellatore.

Lo UML non attribuisce particolare significato semantico all'ordine con cui le corsie di nuoto sono visualizzate. Mentre le azioni possono appartenere a una e una sola swimlane (non a caso si parla di partizioni), è abbastanza normale attendersi transizioni tra azioni appartenenti a diverse swimlane.

Gli oggetti nei diagrammi di attività

Nei diagrammi di attività il concetto di oggetto è presente sia implicitamente, sia esplicitamente. Per quanto riguarda i riferimenti impliciti è sufficiente pensare che i diagrammi di attività sono costituiti, quasi interamente, da stati azione, il cui ciclo di vita coincide con l'esecuzione delle efferenti operazioni. Queste sono implementate da classificatori, e quindi la relativa esecuzione è resa possibile dalle corrispondenti istanze. Le stesse operazioni prevedono un dominio e un codominio spesso costituiti da altri oggetti. Inoltre ogni azione specifica quale sia l'oggetto che ne esegue la relative operazioni. Questi oggetti, qualora si utilizzi la notazione delle swimlane, sono relazionati, in qualche misura, con le corsie di nuoto: tutte le azioni presenti all'interno di una stessa swimlane dovrebbero essere gestite dallo stesso "oggetto" eventualmente complesso.

Chiaramente limitare l'utilizzo degli oggetti ai soli riferimenti impliciti sarebbe troppo restrittivo, e in effetti, la notazione dei diagrammi delle attività prevede la possibilità di visualizzare esplicitamente oggetti che sono input e/o output delle varie azioni. L'elemento utilizzato a tale fine è la relazione di flusso di oggetto.

> Nel metamodello UML questo elemento è rappresentato dalla metaclasse `ObjectFlowState` in cui le frecce entranti e uscenti rappresentano rispettive transizioni (fig. 10.25).

L'elemento flusso di oggetto definisce un flusso tra azioni presenti nei diagrammi di attività. In particolare, sancisce che un'istanza di un particolare classificatore, tipicamente in un

Figura 10.30 — *Diagramma di attività relativo al processo di verifica del PIN da parte di un'ipotetica ATM (autenticazione utente). Nel diagramma non vengono, volutamente, prese in considerazione diverse situazioni di malfunzionamento. Il formalismo dei diagrammi di attività permette di enfatizzare in maniera semplice e efficace processi concorrenti.*

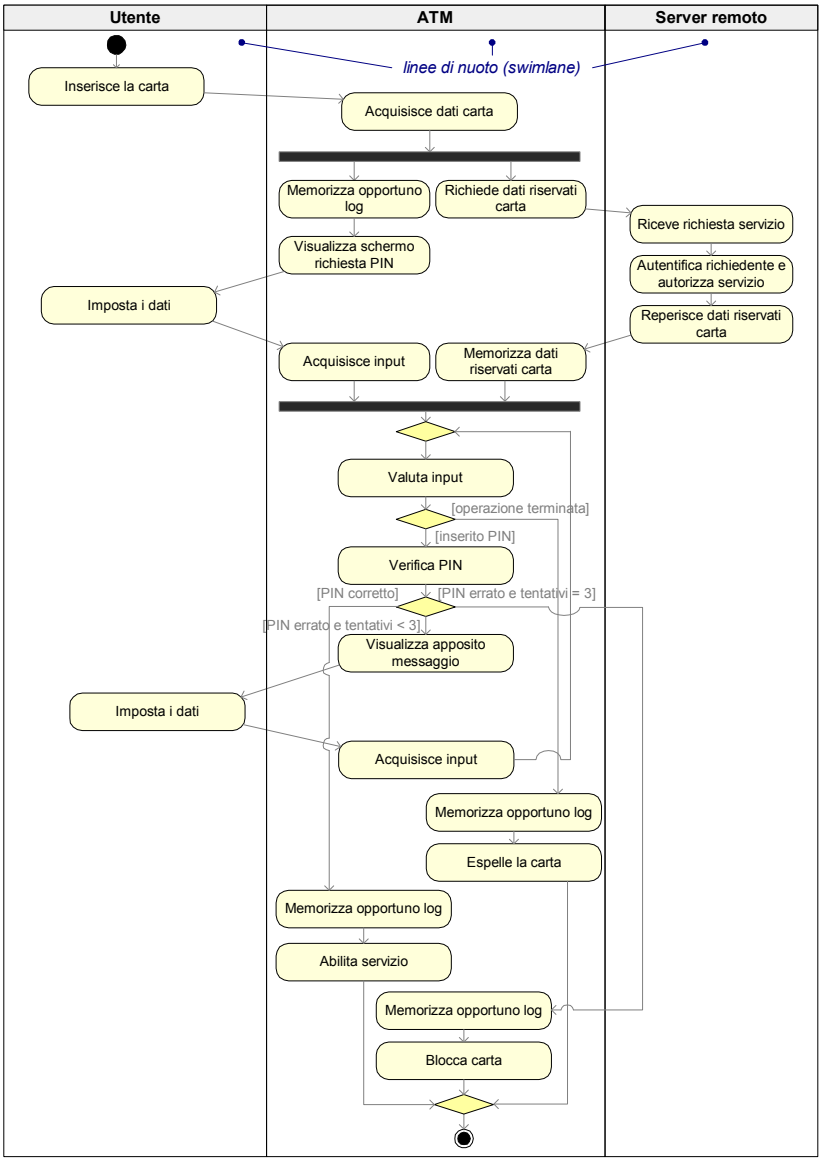
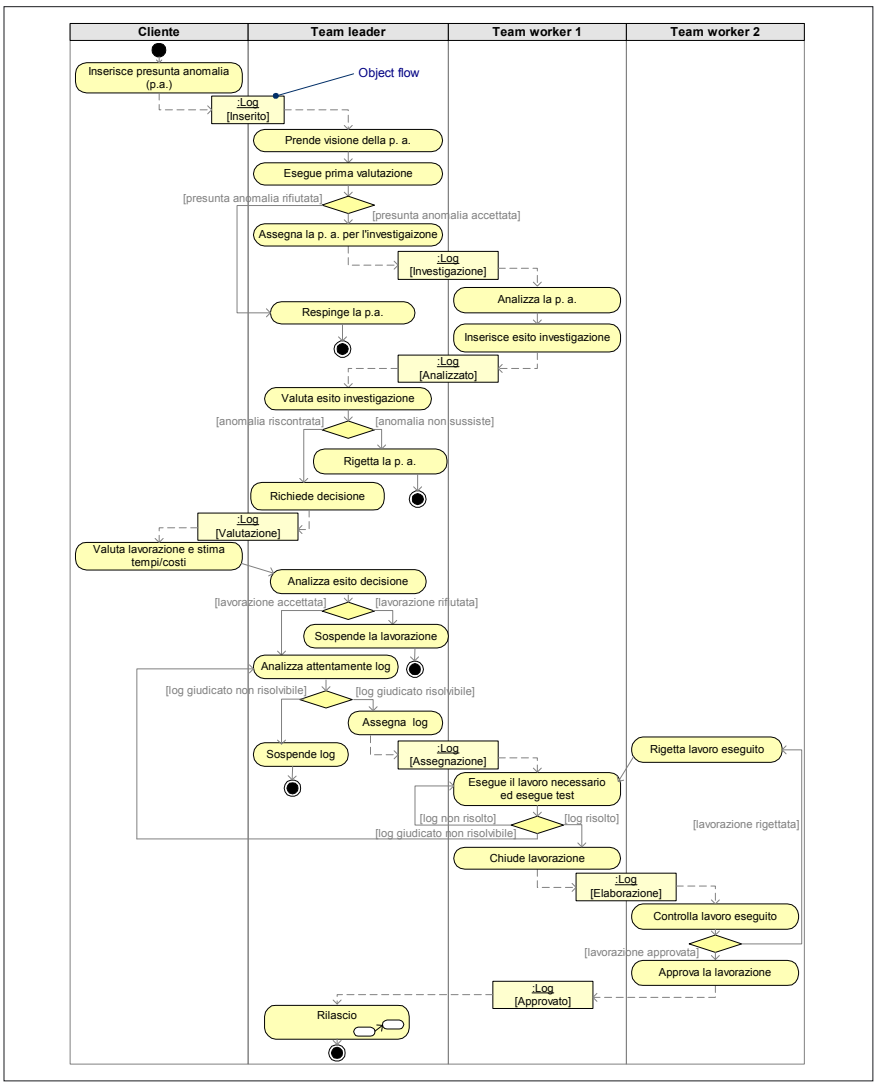


Figura 10.31 — *Diagramma di attività con la procedura business (non è presente una swimlane dedicata al sistema) del diagramma di stato di fig. 10.6. È stata operata una necessaria semplificazione dei flussi e delle swimlane (meglio includere un'ulteriore swimlane per gli elementi del team). Il caso più generale prevede tre elementi: uno incaricato dell'investigazione, uno che effettua la lavorazione vera e propria e l'ultimo deputato alla revisione del lavoro eseguito dal secondo. In sostanza si è preferito conferire più importanza alla notazione dello UML piuttosto che alla presentazione completa della procedura. Nella realtà è fondamentale che i vari diagrammi siano consistenti tra loro, rappresentando la stessa entità da diverse prospettive.*



determinato stato, è disponibile quanto si entra nel relativo flusso di stato. La relazione di flusso di oggetto è visualizzata graficamente attraverso una freccia tratteggiata congiungente lo stato d'azione e il relativo oggetto. A seconda dell'utilizzo dell'oggetto, input o output, la direzione della freccia va, rispettivamente, dall'oggetto al relativo stato di attività o viceversa. Tipicamente, lo stesso oggetto è output di una determinata azione e input di una o più azioni successive. Qualora un oggetto è visualizzato come output di un'azione, la generazione dell'oggetto è innescata dal completamento dell'esecuzione dell'azione. Quando uno stesso oggetto è utilizzato da diverse azioni, tipicamente, queste si occupano di variarne lo stato.

Se si utilizza tale elemento, è il caso di omettere il flusso di controllo, ossia la transizione rappresentata attraverso la linea continua congiungente i due stati tra i quali avviene lo scambio degli oggetti. In effetti, quando uno stato produce un risultato di output utilizzato come input da altri stati, la relazione di flusso di oggetto implica il vincolo di controllo.

Molto spesso gli oggetti inseriti in un diagramma di attività sono manipolati da diverse azioni (altrimenti non avrebbe poi tanto senso rappresentarli). La rappresentazione classica prevederebbe una sola istanza dell'oggetto con molteplici frecce entranti e uscenti. Ma così, oltre a rendere i diagrammi confusi, si rischia di creare ulteriori inconvenienti: si nasconderebbe l'evoluzione dell'oggetto, le comunicazioni tra gli stati, ecc. Per evitare tutti questi inconvenienti, si preferisce mostrare diverse volte, all'interno del medesimo diagramma, lo stesso oggetto e, a ogni apparizione, rappresentarlo in un differente stato del relativo ciclo di vita. Per questo motivo, nelle differenti apparizioni dell'oggetto è consigliato evidenziarne esplicitamente lo stato, racchiuso tra parentesi quadre.

Invio e ricezione dei segnali

I diagrammi di attività, come da prassi UML, definiscono una serie di stereotipi atti a evidenziare particolari specializzazioni e utilizzi degli elementi standard. In particolare, stereotipi molto utili sono quelli utilizzati per evidenziare la ricezione e l'invio di segnali. Si tratta di stati sprovvisti di transizioni interne e di azioni.

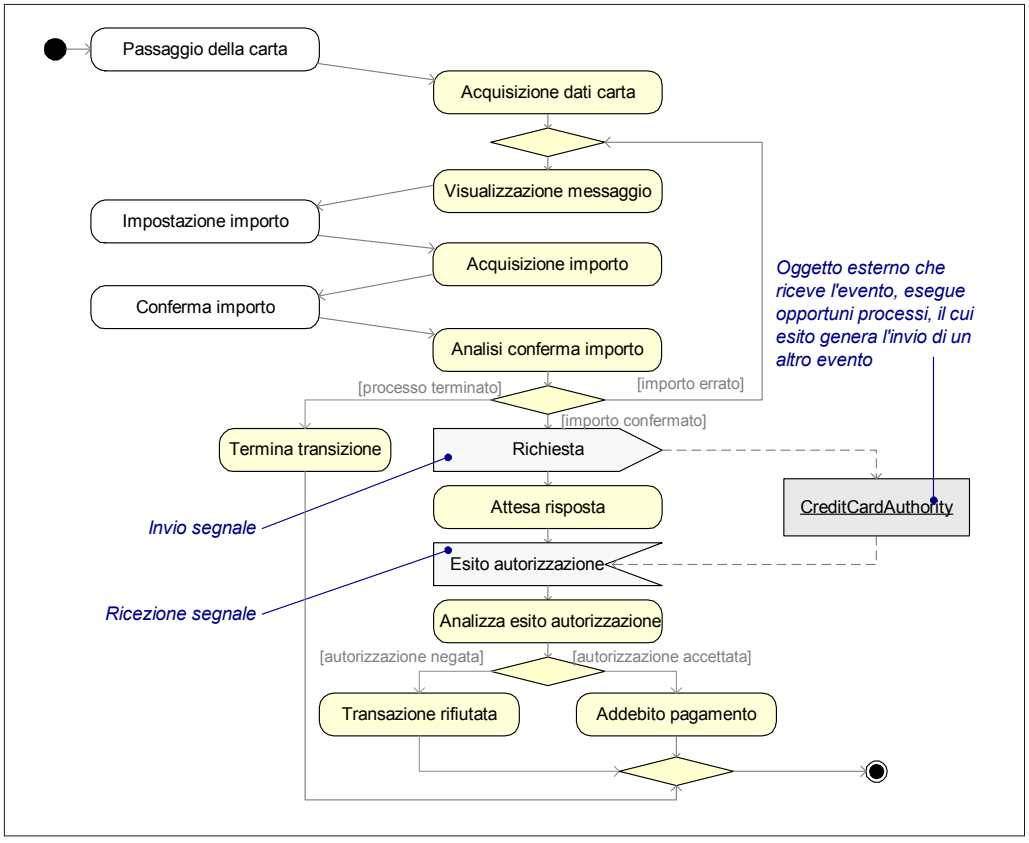
L'elemento relativo alla ricezione di un segnale, questo è visualizzato attraverso un pentagono concavo (la forma è quella di un rettangolo con un triangolo entrante da uno dei suoi lati minori), all'interno del quale è visualizzata la firma del segnale (fig. 10.32).

Questi stati sono inseriti nel grafo di attività attraverso opportune transizioni, non etichettate, che li legano allo stato precedente e a quello successivo (non a quello da cui ha origine il segnale atteso). Volendo, è possibile collegare l'oggetto che origina allo stato di ricezione attraverso una freccia tratteggiata (fig. 10.32).

Per quanto riguarda l'elemento relativo all'invio di segnali, questo è mostrato graficamente attraverso un pentagono convesso con forma speculare all'elemento precedente: un

rettangolo con un triangolo uscente da uno dei suoi lati minori. Come nel caso precedente la firma è riportata all'interno del simbolo (fig. 10.32). L'elemento è connesso nel grafo delle attività attraverso una transizione entrante, non etichettata, proveniente dallo stato precedente, e una uscente diretta verso lo stato successivo. Anche in questo caso è possibile utilizzare una freccia tratteggiata diretta verso l'oggetto dallo stato origine del segnale fino all'oggetto destinatario dello stesso (fig. 10.32).

Figura 10.32 — *Diagramma relativo all'addebito di pagamenti su carta di credito, realizzato per illustrare l'utilizzo dei simboli di ricezione e invio segnale. Come si può notare, il diagramma non fa uso esplicito di swimlane, sebbene sia abbastanza evidente che le azioni di sinistra siano svolte da un commesso, mentre quelle di centro siano a carico del sistema. Molti autori consigliano di utilizzare questa tecnica perché offre tutti i vantaggi delle swimlane, però evita alcuni inconvenienti: maggiore formalità, spazi a disposizione molto ristretti e chiaramente definiti, ecc.*



Eventi differiti

Una situazione che può capitare nella modellazione di diagrammi di stato e di attività consiste nel dover differire la gestione di determinati eventi che scaturiscono mentre una o più attività sono in corso di esecuzione. Si tratta di un problema che, teoricamente, non sarebbe facilmente gestibile: il comportamento normale prevede che qualora un evento non sia gestito immediatamente venga perso.

Si supponga per esempio di modellare un lettore CD e di premere il tasto di espulsione CD mentre lo si sta suonando. In questo caso l'attività "riproduzione del suono" potrebbe differire l'evento "pressione del tasto eject" alla attività di "interruzione riproduzione" che si potrebbe occupare di espellere il CD.

La soluzione utilizzata consiste nell'ipotizzare la presenza di una specifica transizione interna, innescata dall'evento la cui gestione si vuole differire, che si occupi di prelevare tali eventi, immettendoli in un'apposita coda, ove rimangono o fino alla relativa gestione (consumazione dell'evento) o allo scadere di un certo intervallo temporale (l'evento viene effettivamente scartato).

> In realtà, la situazione è risolta direttamente nel metamodello UML attraverso una associazione tra la metaclasse `State` e quella `Event` (che recita il ruolo di evento differito, `deferrableEvent`, fig. 10.15). Questa sancisce che ogni stato può dichiarare una lista di eventi differiti. Quindi si tratta di una caratteristica di particolare importanza, giacché la metaclasse `State` è antenata di diversi elementi (fig. 10.5) utilizzabili come nodi nei diagrammi di stato e di attività.

Ogni stato può dichiarare un insieme di eventi la cui gestione è differita qualora intervengano durante la permanenza nello stato stesso. L'insorgenza di questi eventi, quindi, non innesca alcuna transizione. Se una transizione dipende da un particolare evento e questo è presente nella coda, la transizione è innescata immediatamente a meno della presenza di altri fattori che lo impediscano. Qualora ci si trovi nella condizione in cui potrebbero avere luogo diverse transizioni, viene selezionata quella il cui evento si trova nella posizione più avanzata nella coda: l'evento più vecchio.

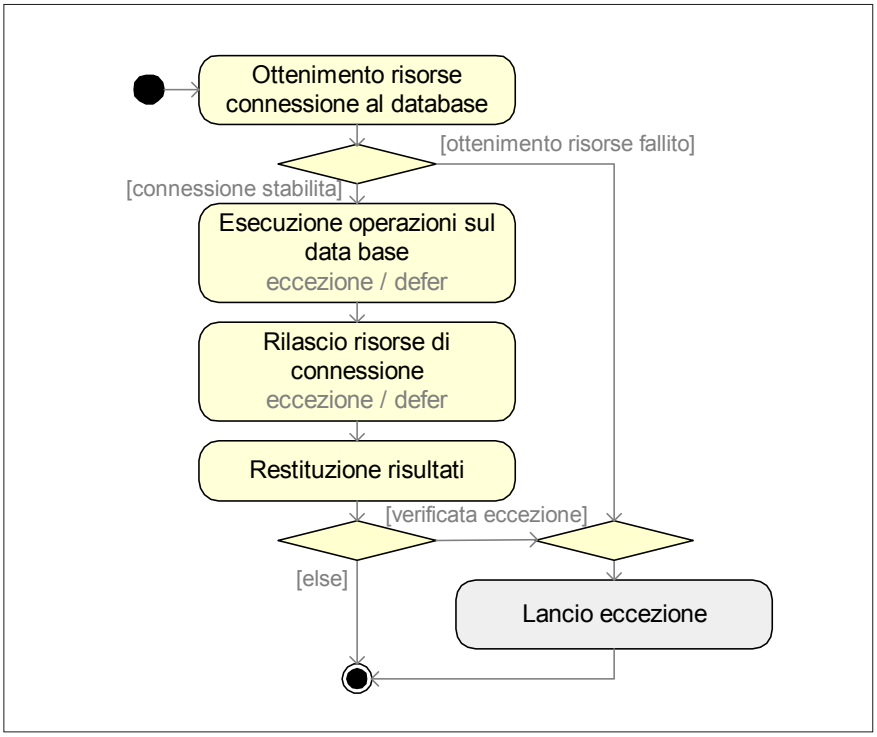
La notazione utilizzata per mostrare gli eventi differiti consiste nel menzionarli in un'apposita lista, con posposte la bara obliqua (`/`) e la parola chiave `deferred`.

Se un evento dichiarato differito all'interno di uno stato dovesse verificarsi, questo viene memorizzato e rilanciato quando si è pronti per transitare nello stato successivo, il quale, a sua volta, può dichiararlo nuovamente differito fino a quanto l'evento giunge a uno stato in cui non è dichiarato differito. In questo caso deve essere gestito o scartato definitivamente. Qualora la dichiarazione di eventi differiti dovesse avvenire all'interno di stati composti o equivalenti (stati di sottomacchina e di sottoattività), il differimento rimane valido per tutti i sottostati. Un'eventualità abbastanza particolare si ha quando uno stato interno generi un evento dichiarato come differito.

Per quanto riguarda gli stati d'azione, è necessario ricordare che l'esecuzione delle relative azioni non è interrompibile per definizione, e pertanto in questi casi potrebbe non avere sempre senso dichiarare eventi differiti. Quindi, sia i normali eventi, sia quelli differiti che potrebbero insorgere durante l'esecuzione dell'azione, devono attendere il termine della stessa prima di poter venir presi in considerazione.

Nel diagramma di fig. 10.33 è presentata, molto schematicamente, un'interazione di un sistema software con una risorsa esterna, quale per esempio un database. Questa interazione (transazione) consta di una serie di passi stabiliti: ottenimento delle risorse necessarie per la connessione, esecuzione delle attività previste e quindi rilascio delle risorse di connessione. Qualora intervenga un'eccezione durante l'esecuzione delle operazioni sul database o anche durante il rilascio delle risorse stesse, è necessario ritardarne la gestione al fine di tentare di rilasciare correttamente le risorse prima di lasciare la sessione di connessione al database. In termini di diagrammi di attività, ciò si traduce dichiarando `deferred` (differito) l'evento di scatenamento di un'eccezione nelle attività di `Esecuzione operazioni sul database` e `Rilascio risorse di connessione`.

Figura 10.33 — *Utilizzo degli eventi differiti.*



Stati di sincronizzazione

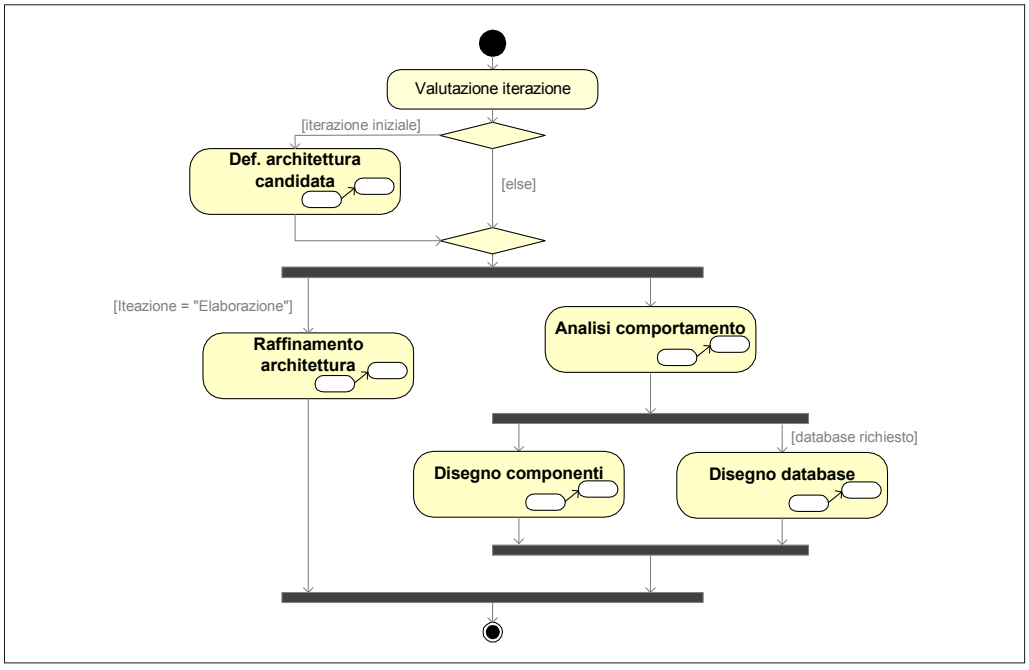
Nei diagrammi di attività, a meno che non si voglia limitare il parallelismo, non è necessario mostrare esplicitamente stati di sincronizzazione. Pertanto è possibile collegare le transizioni concorrenti direttamente a partire dalle barre di sincronizzazione (figg. 10.29 e 10.30).

L'utilizzo degli stati di sincronizzazione è invece necessario qualora, in condizioni di esecuzioni concorrenti, si voglia limitare la differenza tra il numero di transizioni entranti e quello delle transizioni uscenti da una barra di sincronizzazione. In questi casi la notazione rimane equivalente a quanto definito nel paragrafo relativo alle macchine a stati.

I diagrammi di attività permettono di associare condizioni di guardia alle transizioni uscenti da una barra di sincronizzazione utilizzata per la diramazione del controllo (di tipo *fork*). Ciò fa sì che la parte del grafo di attività localizzata a partire dalla transizione uscente dalla fork potrebbe non essere avviata e pertanto non è richiesto che sia completata al punto di ricongiungimento. In tali circostanze, una volta completata l'esecuzione delle restanti diramazioni, è possibile procedere oltre il punto di ricongiungimento.

Si consideri il diagramma di fig. 10.34. Esso rappresenta le macroattività da effettuarsi durante la fase di analisi e disegno secondo il processo RUP. Come si può notare, è

Figura 10.34 — *Rappresentazione del workflow di analisi e disegno. Questo diagramma illustra le macroattività da svolgere durante tale fase secondo i dettami del processo RUP.*



necessario eseguire le attività di raffinamento dell'architettura solo nelle fasi di *elaborazione* (nelle precedenti l'architettura non dovrebbe essere ancora definita); così come l'attività di disegno del database va eseguita solo nel caso in cui il sistema ne preveda la necessità (in sostanza nel 99% dei casi). Qualora questa attività non venga eseguita, è possibile terminare la regione successiva all'analisi del comportamento, subito dopo aver disegnato i componenti del sistema e ovviamente senza attendere il disegno del database stesso. Nella realtà si verifica che le attività di scomposizione del sistema in componenti e quella del disegno del database siano intimamente interdipendenti.

Invocazioni dinamiche

Le azioni specificate in uno stato d'azione o nel grafo di attività presente in uno stato di sottoattività possono essere eseguite diverse volte in maniera concorrente. Il numero delle invocazioni concorrenti è stabilito a tempo di esecuzione da un'espressione di concorrenza, la quale prevede come dominio una lista di argomenti, in cui ogni elemento è relativo a una singola invocazione.

Per quanto concerne la notazione, qualora la concorrenza dinamica di un'azione assuma un valore diverso dall'unità, la relativa molteplicità è mostrata nell'angolo in alto a destra, altrimenti viene omessa.

> Nel metamodello, le proprietà relative alle invocazioni dinamiche sono rappresentate attraverso opportuni attributi dichiarati nelle classi `ActionState` e `SubactivityState` (fig. 10.25). Questi attributi (non mostrati in fig. 10.25 per problemi grafici) sono:
>
> - `isDynamic`: valore booleano che permette di discernere invocazioni concorrenti dalle altre. Un valore `false` rende non significativi i successivi attributi;
> - `dynamicMultiplicity`: valore di molteplicità che permette di limitare il numero di esecuzioni parallele;
> - `dynamicArguments`: espressione la cui valutazione genera una lista di oggetti (`ArgListsExpression`) che determina, a tempo di esecuzione, il numero di esecuzioni parallele delle azioni presenti nello stato. Ogni valore deve essere una lista di oggetti, utilizzata come parametro in input per una specifica esecuzione.

Utilizzo

In merito all'utilizzo dei diagrammi di attività si è avuto modo di spendere molte pagine nel corso di tutto il testo. In particolare la discendenza diretta dai famosi diagrammi di flusso, nonché dai DFD (*Data Flow Diagram*) ne rende l'utilizzo quasi universale… Più o meno tutti si sono trovati, in qualche modo, di fronte a un diagramma di flusso.

L'immediatezza e la familiarità dell'utilizzo (anche se poi per un impiego formale le cose si complicano) ne costituiscono le caratteristiche peculiari che lo rendono popolare a tutti, anche agli utenti con pochissima esperienza informatica. Ciò spesso si tramuta,

paradossalmente, in difetto: non appena gli utenti cominciano a saper leggere qualche diagramma, immediatamente si sentono provetti modellatori, e niente sembra lontano dalla loro portata. Purtroppo poi gli esiti non sono esattamente quelli sperati.

I diagrammi delle attività si prestano a essere impiegati per miriadi di funzioni: analisi dei requisiti utenti, modellazione dei workflow previsti dai processi di sviluppo del software (fig. 10.29), illustrazione di particolari meccanismi/procedure (fig. 10.30), disegno di algoritmi (fig. 10.26 e 28), specie paralleli, ecc.

Per quanto concerne il ciclo di vita del software, l'utilizzo di questi diagrammi va dall'analisi dei requisiti fino al modello di disegno, concentrandosi proprio in queste due fasi così distanti.

Durante l'analisi dei requisiti (*cfr* Capitolo 5) si configurano come uno strumento validissimo per la cattura dei requisiti: si tratta di un formalismo grafico non estraneo agli utenti che permette di rappresentare formalmente regole del business, ecc. In particolare, permettono di instaurare una magnifica piattaforma di colloquio con gli utenti, non senza effetti collaterali, come aumento del tempo di produzione e manutenzione dei diagrammi. In questo contesto possono essere utilizzati sia per descrivere gli scenari di un solo caso d'uso, sia per descrivere la collaborazione tra diversi. In quest'ultimo caso sono impiegati per modellare processi di business (fig. 10.31). Sempre in questa fase si prestano a descrivere formalmente business rule particolarmente complesse. Per ulteriori dettagli, far riferimento al Capitolo 5.

Nel corso della fase di analisi, l'utilizzo è piuttosto limitato, per poi riprendere quota durante la fase di disegno. In questo ambito sono utilizzati per comunicare formalmente specifici algoritmi, specie se concorrenti. La notazione dei diagrammi di attività sembrerebbe particolarmente vantaggiosa per questo tipo di utilizzo.

Un'interessante considerazione da farsi è che, se un algoritmo necessita di essere realizzato attraverso un diagramma, significa che la sua complessità non è trascurabile, pertanto la modellazione si configura come un ottimo strumento per dar luogo a opportune analisi e revisioni dello stesso. Inoltre, prima di concentrarsi nella stesura di procedure in codice particolarmente complesse, verosimilmente è una buona idea investire qualche ora nel tracciare un diagramma, al fine di investigarne la validità, correggere eventuali errori ed eseguire specifiche ottimizzazioni. Ciò, oltre a fornire un'ottima documentazione, permette di eseguire i suddetti processi in modo visivo, invece che in termini di codice, attività che, molto spesso, non è assolutamente agevole…

Questa frase probabilmente non suscita le gioie dei "puristi" dell'XP per i quali tutte le attività devono vertere sul codice.

Stile

Introduzione

Come si è ripetuto diverse volte, la notazione dei diagrammi di attività rappresenta una particolare variante dei diagrammi di stato: è pertanto lecito attendersi che molte delle linee guida illustrate per i diagrammi di stato mantengano la loro validità (con minime variazioni) nel contesto dei diagrammi di attività.

In particolare le linee guida che è possibile applicare, con minimo impegno, nel contesto dei diagrammi di attività sono:

- "2.5.5. Sindacare l'assenza di pseudostati iniziali e degli stati finali". Questi elementi sono particolarmente importanti nel contesto dei diagrammi di attività. La relativa assenza molto raramente è giustificabile;
- "2.5.6. Curare la posizione degli pseudostati iniziali e degli stati finali";
- "2.5.7. Controllare i buchi neri e gli 'stati dei miracoli' ".

> A dire il vero, esistono altre linee guida che mantengono la loro validità, ma l'applicazione nel contesto dei diagrammi di attività non è così immediata e quindi si è deciso di riportare esplicitamente il relativo paragrafo.

Punti decisionali

Nei diagrammi di attività i punti decisionali (essenzialmente per questioni di retaggio storico) sono rappresentati attraverso il simbolo del rombo. Però, a differenza delle notazioni passate, quali per esempio quella dei Flow Chart, non è possibile includervi la condizione che dà luogo alla diramazione del flusso. Questo perché nello UML i punti decisionali sono istanze di pseudostati utilizzati per collegare transizioni composte. Quindi, ogni punto di decisione deve prevedere una transizione proveniente da un apposito stato nel quale si esegue l'azione i cui risultati possono generare la diramazione. Molto importante è assicurarsi che ogni transizione uscente da un punto di decisione sia corredata da apposita condizione di guardia, che specifica la clausola che ne abilita la percorrenza. Per esempio nel diagramma di fig. 10.32, a seguito della conferma dell'importo da parte del commesso, è necessario specificare un'apposita attività di analisi dei dati impostati prima di poter eseguire la diramazione del flusso del controllo.

Chiaramente non sempre è necessario e opportuno riportare tutte le possibili condizioni di guardia, spesso è possibile semplificare il numero di transizioni uscenti ricorrendo alla parola chiave `else` (*cfr Diagrammi di attività. Elementi di decisione e di ricongiunzione*).

Un'altra considerazione da operare è che, nei diagrammi di attività, non sempre è necessario riportare esplicitamente i punti decisionali; questi possono essere rappresentati da transizioni, corredate da opportune condizioni di guardia, uscenti dallo stato in cui si

esegue l'azione che può dar luogo alla diramazione. Pertanto, nei casi in cui i punti di decisione risultino superflui, probabilmente è opportuno valutare bene la necessità di inserirli.

Analizzare attentamente le condizioni di guardia

Come riportato nel paragrafo precedente, le condizioni di guardia devono essere presenti in ogni transizione uscente dai punti decisionali, ed è necessario valutare l'opportunità di utilizzare la parola chiave `else` per semplificare il numero di transizioni uscenti.

Inoltre, come nel caso dei diagrammi di attività, è opportuno evitare condizioni di guardia sovrapposte, come per esempio `[tentativi <= max]` e `[tentativi >= max]`. In queste condizioni non è prevedibile la selezione della transizione qualora si verifichi il caso in cui `tentativi = max`.

Importante è anche porre attenzione alla situazione opposta, ossia al caso in cui le condizioni di guardia possano dar luogo a condizioni non gestite come, per esempio, `[tentativi < max]` e `[tentativi > max]`. Nuovamente si genera la situazione in cui non è specificato cosa avvenga qualora si verifichi la situazione `tentativi = max`.

Accoppiare le barre di sincronizzazione

I diagrammi di attività permettono di specificare azioni concorrenti per mezzo delle barre di sincronizzazione (in maniera del tutto analoga ai diagrammi di attività). In particolare, si distinguono le barre che permettono la diramazione del flusso di controllo (`fork`) e quelle che permettono il ricongiungimento (`join`). Chiaramente le prime devono precedere le seconde. Le barre di diramazione del controllo sono caratterizzate dal possedere una transizione entrante e almeno due uscenti, mentre per quelle di ricongiungimento si ha la configurazione opposta: almeno due transizioni entranti e una sola uscente.

Molto importante è assicurarsi che a ogni diramazione del flusso corrisponda un ricongiungimento, e che — a meno di casi particolarissimi, come quelli dovuti a stati composti concorrenti — non siano presenti flussi generati da una diramazione che non confluiscano in alcuna barra di ricongiungimento (figg. 10.29 e 10.30).

Utilizzare le swimlane per migliorare la qualità del diagramma

Le swimlane permettono di partizionare il diagramma di attività in opportuni sottoinsiemi disgiunti di stati. Una tecnica conveniente consiste nel far corrispondere le linee di guida a particolari entità al fine di enfatizzare l'assegnazione delle responsabilità. Per esempio, in fase di analisi dei requisiti, è consigliabile utilizzare le *corsie di nuoto* per evidenziare l'allocazione delle responsabilità (esecuzione delle attività) alle varie parti in causa. Queste sono costituite da attori, unità dell'organizzazione ecc.

Qualora si utilizzi la notazione dei diagrammi di attività per illustrare determinati algoritmi, tipicamente non ha molto senso utilizzare le swimlane, in quanto tutte le attivi-

tà sono a carico di un'unica entità: il sistema. Per questa regola esistono delle eccezioni. Per esempio, potrebbe avere senso utilizzarle qualora l'esecuzione sia suddivisa in più risorse, oppure nel caso in cui l'esecuzione avvenga attraverso più strati dell'architettura. In questi casi sarebbe abbastanza naturale associare le swimlane, rispettivamente, alle diverse risorse e ai vari strati dell'architettura.

Studiare la posizione delle swimlane.

Le specifiche ufficiali dello UML, non assegnano alcun particolare significato semantico alla posizione delle swimlane, che quindi è completamente demandata al modellatore.

Si tratta di elementi che hanno un notevole impatto sulla qualità dell'intero diagramma. Analizzando i vari modelli è abbastanza naturale evidenziare delle partizioni che danno luogo a un'intensa interazione, da altre in cui questa è nulla. È abbastanza naturale quindi disporre contiguamente swimlane in cui l'interazione è significativa.

Qualora si utilizzi la notazione dei diagrammi di attività per evidenziare servizi offerti dal sistema, una tecnica che l'autore del presente libro tende a utilizzare consiste nel posizionare le swimlane al fine di evidenziare l'architettura del sistema, un po' come suggerito nel contesto dei diagrammi di interazione. Pertanto si inizia con le swimlane relative agli attori posti a sinistra, poi si prosegue con quelle relative alle istanze dei moduli di interfaccia utente, quella dello strato dei servizi business, quella degli oggetti business e lo strato di integrazione. In virtù di questa organizzazione, è logico attendersi che istanze appartenenti a strati adiacenti diano luogo a un certo scambio di messaggi, che invece è nullo tra elementi appartenenti a strati non adiacenti.

Qualora invece si utilizzi la notazione dei diagrammi di attività per modellare le dinamiche di particolari casi d'uso, è possibile utilizzare un approccio simile, inserendo le swimlane relative agli attori a sinistra, quelle attinenti al sistema o a sue parti a destra. Nel caso in cui un'interazione preveda la partecipazione di sistemi esterni che non avviano l'erogazione del servizio, è opportuno riportare questi ultimi all'estrema destra del diagramma. Medesimo accorgimento può essere riservato per eventuali attori secondari (che quindi ricevono unicamente notifiche dal sistema).

Limitare il numero di swimlane.

Molto spesso, analizzando diagrammi di attività è possibile evidenziare la presenza di molteplici *swimlane*. Da tener presente che qualora queste superino le 4-5 unità, il diagramma tende a divenire meno fruibile. È il caso, forse, di ricorrere a sottostati di attività.

Quando poi non si possa veramente fare a meno di dover presentare diverse swimlane, ognuna delle quali possegga un numero piuttosto contenuto di azioni, potrebbe valere la pena di rappresentare le swimlane orizzontalmente. Il problema è che si tratta di un approccio un po' desueto che ben pochi tool permettono.

Una tecnica alternativa consiste nel raggruppare i vari elementi come se esistessero swimlane, ma senza rappresentarle (fig. 10.32), con un maggiore grado di libertà e un'ottimizzazione degli spazi.

Selezionare opportunamente i flussi di oggetto

La notazione dei diagrammi di attività permette di rappresentare esplicitamente gli oggetti utilizzati (input e/o output) da determinate operazioni. Il primo consiglio è quello di non rappresentare esplicitamente tutti gli oggetti manipolati all'interno di uno stesso diagramma. Una tecnica siffatta tenderebbe a produrre diagrammi altamente caotici. Molto importante è quindi riportare unicamente gli oggetti la cui visualizzazione concorra effettivamente a aumentare il contenuto informativo del diagramma.

Una seconda constatazione è relativa alla visualizzazione di tali oggetti. Nella quasi totalità dei casi, visualizzare una sola volta un determinato oggetto, con molte linee entranti e uscente, non solo renderebbe il diagramma confuso ma, verosimilmente, renderebbe meno espliciti importanti flussi. Pertanto è consigliabile riportare uno stesso oggetto più volte, evidenziando, di volta in volta, lo stato in cui l'azione a cui è associato come output lo ha fatto transitare. Questi stati, riportati esplicitamente racchiusi tra parentesi quadre, devono essere utilizzati consistentemente e quindi devono essere prelevati dal relativo diagramma di stato.

Studiare la posizione dei flussi di oggetto

Qualora si utilizzino le swimlane e contestualmente con la rappresentazione degli oggetti nei diversi stati, viene abbastanza naturale posizionare l'oggetto, se possibile, a cavallo delle swimlane in cui sono presenti le azioni che lo prevedono, rispettivamente, come output e come input.

Organizzare gerarchicamente i vari diagrammi

Se si devono modellare algoritmi o operazioni particolarmente complesse, probabilmente è opportuno organizzare gerarchicamente i diagrammi secondo dettami della mitica strategia top-down. Si realizza una versione del diagramma con macro attività (o meglio sfruttando stati di sottoattività) demandando ad altre parti la relativa definizione di dettaglio. La quasi totalità dei tool moderni agevola tale tecnica permettendo di collegare tra loro diversi diagrammi, attraverso opportuni link percorribili a colpi di click del mouse. Questa tecnica permette di semplificare i diagrammi e di rendere possibile il concentrarsi solo sugli aspetti più significativi in un determinato momento.

Ricapitolando…

Il presente capitolo è stato dedicato all'illustrazione della notazione dei diagrammi delle carte di stato (*statechart diagrams*) e di attività (*activity diagrams*), con la quale si è ultimata la descrizione, avviata nel capitolo precedente, dei formalismi forniti dallo UML per modellare gli aspetti dinamici di un sistema. In particolare, i diagrammi delle carte di stato sono utilizzati per mostrare il ciclo di vita di entità quali istanze di una classe, casi d'uso, il sistema nel suo complesso, ecc.; mentre i diagrammi di attività sono utilizzati principalmente per modellare workflow, ossia flussi di lavoro (processi) rappresentati da un insieme di attività, e dagli oggetti prodotti durante il processo stesso. Entrambe le notazioni prevedono un utilizzo piuttosto ampio, abbracciando l'intero processo di sviluppo del software: dall'analisi dei requisiti al disegno del sistema.

I diagrammi di stato sono utilizzati per modellare il ciclo di vita delle istanze di un elemento di un modello: permettono di rappresentare la possibile sequenza di stati e di azioni attraverso le quali le istanze dell'elemento considerato possono transitare durante il proprio ciclo di vita. Ogni transizione, tipicamente, è prodotta delle operazioni eseguite per reazione agli eventi ricevuti. Esempi di eventi sono segnali, il trascorrere di un determinato lasso di tempo, l'invocazione di un'operazione, ecc.

I diagrammi di stato affondano le proprie radici, nella teoria degli automi a stati finiti, la cui definizione accademica sancisce che un Automa a Stati Finiti (FSA) è il modello più semplice di dispositivo computazionale, formato da un processore centrale di capacità finita basato sul concetto di stato. A sua volta, il FSA si divide in deterministico (DFSA) e non deterministico (NDFSA). Un DFSA viene definito come una quintupla

$<Q, I, t, q_0, F>$

dove Q è un insieme finito di stati; I è un insieme finito di segnali di input; t è la funzione di transizione $t : \{Q \times I\} \rightarrow Q$ che "trasforma" un elemento del dominio (insieme degli input) in un elemento del codominio (insieme degli output); q_0 è lo stato iniziale, $q_0 \in Q$; F è l'insieme degli stati finali, o di accettazione; $F \subseteq Q$.

La notazione dei diagrammi delle carte di stato è utilizzata prevalentemente per rappresentare automi a stati finiti, con diverse differenze. Dal punto di vista della notazione, un diagramma è un grafo costituito da vertici e archi che li connettono. In questo ambito, i vertici del grafo sono gli stati (e altri elementi similari come gli pseudostati), mentre le connessioni tra questi, tipicamente, rappresentano le transizioni di stato. Uno stato rappresenta una condizione durante il ciclo di vita di un oggetto, o di un'interazione, durante la quale questo soddisfa determinate condizioni, esegue specifiche operazioni e attende l'occorrenza di opportuni eventi.

Gli stati sono mostrati graficamente attraverso rettangoli dagli angoli smussati, eventualmente divisi in diversi compartimenti. Il primo, opzionale, è utilizzato per rappresentare il nome dello stato. Uno stato senza compartimento relativo al nome è detto anonimo. Qualora nello stesso diagramma siano rappresentati diversi stati anonimi, questi vanno considerati distinti. Il secondo compartimento è relativo alle transizioni interne (*internal transitions*), utilizzato per mostrare una lista di azioni o attività interne da eseguire quando l'istanza si trova in tale stato e si verifichino particolari eventi. Le transizioni interne sono del tutto equivalenti alle auto transizioni (*self transition*) eccetto per il fatto che lo stato non termina e quindi non si generano gli eventi di uscita e di rientro nello stesso. Il formato di una transizione interna prevede la seguente configurazione: `event-name '(' comma-separated-parameter-list ')' '[' guard-condition ']' '/' action-expression`.

Lo UML prevede i seguenti eventi standard da utilizzarsi per le transizioni interne: `entry` (innescato all'entrata dello stato), `exit` (innescato all'uscita dello stato), `do` (innescato durante la permanenza nello stato) e `include` (utilizzato per includere altri diagrammi degli stati).

Gli stati possono essere di tre tipi fondamentali: semplice, finale e composto. Uno stato semplice è caratterizzato dal non prevedere un'ulteriore decomposizione e pertanto è antitetico allo stato composto che invece la prevede. Lo stato finale sancisce il termine dell'esecuzione della relativa macchina a stati.

Tra i vari pseudostati, uno di particolare importanza è quello iniziale, che indica la creazione dell'istanza dell'elemento di cui il diagramma modella il ciclo di vita, e quindi identifica l'avvio della macchina a stati. Graficamente è rappresentato attraverso una piccola circonferenza piena. La transizione da uno pseudostato iniziale può essere etichettata con l'evento che genera l'oggetto, di cui si rappresenta il ciclo di vita.

Lo stato finale indica che lo stato composto che lo ospita ha terminato il proprio ciclo di vita. Qualora questo stato è quello di primo livello, significa che l'esecuzione dell'intera macchina a stati è terminata. Graficamente è rappresentato attraverso due piccole circonferenze concentriche, di cui quella interna piena (la classica configurazione denominata "a occhio di bue").

Uno stato è detto composto quando può essere ulteriormente definito per mezzo di una macchina a stati. Di stati composti esistono due tipologie: quella che prevede la relativa definizione per mezzo di due o più macchine a stati concorrenti e quella in cui vi siano semplicemente un insieme di stati connessi. La notazione grafica standard degli stati composti prevede un'ulteriore sezione, rispetto a quelle degli stati semplici, dedicata all'illustrazione della struttura della macchina a stati interna. Una transizione terminante in uno stato composto equivale a una transizione diretta verso il relativo pseudostato iniziale, il quale deve essere esplicitamente specificato. Eventualmente è possibile specificare transizioni dirette verso uno specifico stato interno, eventualmente annidato. Una transizione terminante allo stato finale di uno stato composto non concorrente ne determina il completamento, e quindi innesca l'evento di uscita dallo stato. Analogamente alle transizioni di entrata, anche quelle di uscita possono lasciare un determinato stato interno.

Qualora uno stato composto sia costituito da sottostati concorrenti, questi sono mostrati suddividendo la sezione dedicata alla macchina a stati interni, in diverse zone orizzontali denominate regioni, evidenziate attraverso opportune linee tratteggiate.

Una transizione diretta verso uno stato composto, dotato di sottostati concorrenti, corrisponde a innescare le transizioni uscenti da tutti gli pseudostati iniziali presenti nelle varie regioni. Il completamento dello stato è raggiunto a seguito del completamento delle attività in tutte le regioni concorrenti. Una transizione diretta a uno specifico stato di una particolare regione genera, automaticamente, l'innesco delle transizioni di uscita dagli pseudostati iniziali presenti nelle altre regioni. Una transizione uscente dallo stato composto, relativa al verificarsi di un particolare evento, forza la terminazione di tutte le regioni di cui si compone, le quali eseguono l'eventuali azioni associate con l'evento di uscita, quindi le azioni associate alla transizione possono aver luogo. Alla fine delle esecuzione di tali azioni, si transita nel nuovo stato.

Talune volte, per comodità, è possibile rappresentare uno stato composto attraverso la notazione sintetica al fine di ridurre la complessità dei diagrammi, di rappresentare più agevolmente diagrammi complessi, ecc. Questa notazione prevede di non rappresentare il dettaglio interno dello stato, mostrando al suo posto un'apposita icona (due cerchietti uniti da un segmento). La notazione sintetica però può generare diversi problemi di rappresentazione qualora vi siano transizioni destinate o originate a uno stato interno allo stato composto. Per

risolvere questo problema, è possibile mostrare queste transizioni come destinate o originate da appositi stub facenti funzione degli stati non visualizzati. Questi stub sono rappresentati graficamente attraverso sottili segmenti orizzontali (barrette) visualizzati all'interno dell'elemento rappresentante lo stato composto.

Un indicatore di stato storico (*history state indicator*, più semplicemente detto stato storico), è mostrato graficamente attraverso la lettera `H` circondata da un cerchietto. Questo stato può avere diverse transizioni entranti, mentre deve prevederne una sola uscente, terminante nello stato a cui si passa per default, qualora non si sia mai entrati nella regione di appartenenza dello stato storico. Il comportamento a regime, prevede che, una volta raggiunto lo stato storico, questo inneschi l'unica transizione uscente, la quale transita nell'ultimo stato dello stato composto, in cui la macchina si trovava prima dell'ultima uscita da questo. In alcuni contesti è necessario disporre di uno stato storico profondo, il quale sia in grado di riprendere l'esecuzione di un sottostato, a qualsiasi livello di annidamento, in cui si trovava la macchina a stati prima dell'uscita dallo stato composto. La differenza dallo stato storico semplice è pertanto che non si restringe l'attenzione ai soli stati dello stesso livello dell'indicatore dello stato storico. In questo caso la notazione grafica prevede un asterisco posposto alla lettera `H` (`H*`).

Un evento è definito come un avvenimento di una certa importanza. Nell'ambito dei diagrammi degli stati, si restringe l'attenzione ai soli eventi in grado di innescare una transizione di stato. In particolare, in questo contesto, si possono presentare nelle seguenti forme:

1. una condizione booleana, stabilita a tempo di disegno che, assumendo il valore vero (`true`) dà luogo a un'istanza di tipo evento di cambiamento;

2. la ricezione di un segnale esplicito inviato da un oggetto a un altro. In questo caso si ha un'istanza della classe evento segnale;

3. la ricezione, da parte di un oggetto, di un'invocazione di un'operazione implementata come una transizione. In questo caso si ha un'istanza di tipo evento di chiamata;

4. lo scadere di un definito intervallo di tempo, dopo la ricezione di un determinato evento, come per esempio l'entrata di uno stato concorrente, oppure il raggiungimento di una determinata data e orario. In questi casi si ha un'istanza di evento temporizzato.

La notazione utilizzata per gli eventi segnale e di chiamata, prevede una stringa del seguente formato: `event-name '(' comma-separated-parameter-list ')'` dove la stringa `comma-separated-parameter-list` ha il formato: `parameter-name : type-expression.` Un evento temporizzato è specificato per mezzo della parola chiave `after` seguita dall'espressione da valutare, a tempo di modellazione, indicante l'intervallo di tempo specificato. Qualora l'evento temporizzato debba avvenire in un determinato istante di tempo, questo può essere dichiarato dopo la parola chiave `when`.

Una transizione semplice è una relazione tra due stati indicante che un'istanza nel primo stato entrerà nel secondo, eseguendo determinate azioni, al verificarsi di un prestabilito evento, qualora le eventuali condizioni specificate (condizioni di guardia) siano specificate.

Gli eventi sono processati uno alla volta e vengono ignorati qualora non diano luogo ad alcuna transizione. La situazione opposta, cioè la possibilità che lo stesso evento inneschi diverse transizioni nel contesto di una stessa regione sequenziale, prevede che solo una sia scatenata. La notazione utilizzata per le transizioni è, chiaramente, la stessa riportata nella sezione delle transizioni interne, pertanto assume il formato: `event-name '(' comma-separated-parameter-list ')' '[' guard-condition ']' '/' action-expression`.

Una transizione è definita concorrente qualora dia luogo a una sincronizzazione di due o più transizioni originate da diversi stati concorrenti e/o, caso opposto, generi una suddivisione del flusso di controllo in più flussi concorrenti destinati a due o più stati concorrenti.

La notazione grafica prevede l'utilizzo di un segmento visualizzato con tratto spesso, denominato barra di sincronizzazione (*synchronization bar*). Questa può essere utilizzata per rappresentare sincronizzazioni, suddivisioni del flusso di controllo o entrambe.

Le transizioni composte sono percorsi di transizioni, connesse attraverso pseudostati transienti. Queste transizioni permettono di realizzare segmenti di transizioni condivise da diversi percorsi. Un esempio classico è fornito da un singolo evento che può terminare in un insieme di stati differenti e la selezione dipende dal risultato dell'esecuzione di un'azione eseguita dopo l'innesco di una transizione composta. Le transizioni composte si prestano a essere rappresentate graficamente mediante i punti di congiunzione (*junction point*). Per quanto concerne la rappresentazione grafica, i punti di congiunzione sono rappresentati, nei diagrammi degli stati, per mezzo di piccole circonferenze nere. I punti di congiunzione si dividono in due categorie: punti di diramazione statica (*static branch point*) e punti di selezione dinamica (*dynamic choice point*). I primi sono caratterizzati dal fatto che le transizioni uscenti dal punto di congiunzione possiedono delle condizioni di guardia, tipicamente a mutua esclusione, il cui esito può essere stabilito fin dal momento in cui si innesca la transizione che porta al punto di congiunzione. I secondi sono utilizzati per modellare decisioni dinamiche: solo dopo aver innescato la transizione diretta a tale punto è possibile valutare le condizioni di guardia delle transizioni uscenti e quindi solo allora è possibile selezionare la transizione di uscita. I punti di selezione dinamica sono rappresentati graficamente attraverso una circonferenza vuota.

Uno stato di sincronizzazione (s*ynch state*) è utilizzato per sincronizzare macchine a stati (concorrenti) di diverse regioni di uno stato composto concorrente. Il suo utilizzo avviene in congiunzione con le barre di sincronizzazione. Questo accorgimento è necessario per assicurarsi che la macchina a stati di una particolare regione lasci uno specifico stato, prima che la macchina a stati di un'altra regione entri in un determinato stato. Gli stati di sincronizzazione sono visualizzati graficamente attraverso una circonferenza con specificato all'interno un numero positivo indicante la differenza massima consentita tra il numero delle transizioni che lasciano lo stato di sincronizzazione e quelle che entrano. Nel caso in cui non si voglia imporre alcun limite, si utilizza il carattere asterisco.

Uno stato di sottomacchina (*submachine state*) rappresenta l'invocazione di una macchina a stati definita da qualche altra parte. In generale, è possibile accedere a un qualsiasi sottostato di una macchina a stati invocata, quantunque, l'accesso preferenziale è quello relativo allo pseudostato iniziale. Analogamente, è possibile uscire da qualsiasi sottostato di una macchina invocata, sebbene il caso tipico è quello della stessa macchina che raggiunge il proprio stato finale. Entrate e uscite non standard vengono visualizzate per mezzo di speciali stati stub.

Dal punto di vista della rappresentazione grafica, lo stato di sottomacchina è rappresentato utilizzando la notazione classica, con in più, nel compartimento dedicato alle transizioni interne, la dichiarazione di inclusione (`include`), che permette di dichiarare la sottomacchina a stati invocata.

I diagrammi delle macchine a stati sono una notazione dello UML particolarmente utile e potente atta a rappresentare particolari aspetti dinamici del sistema. Nel contesto dei processi formali di sviluppo del software sono utilizzabili praticamente in tutti gli stadi. Durante la fase di analisi dei requisiti, è conveniente utilizzare la notazione per definire in maniera semplice, formale e intuitiva, il ciclo di vita di oggetti business particolarmente importanti. Un altro utilizzo consigliato consiste invece nel rappresentare il ciclo di vita di particolari parti del modello di disegno, come determinati *engine* o parser.

La prima linea guida inerente lo stile è riferita ai nomi degli stati. In particolare, si consiglia di utilizzare nomi semplici, brevi e descrittivi. Per ciò che riguarda i nomi delle transizioni dovrebbero essere costituiti da verbi che richiamino l'evento che le ha generate. Chiaramente anche la posizione di questi elementi è molto importante: è opportuno enfatizzare i nomi degli stati e porre le transizioni vicino agli stati sorgenti. Molto importante è poi evitare il ricorso a stati anonimi, quantunque sia permesso dalla notazione.

Tutte le macchine a stati di primo livello, non incorporate in altre macchine a stati, devono necessariamente prevedere uno pseudostato iniziale e uno stato finale. Molti autori consigliano di sistemarli in modo tale da favorire una lettura dei diagrammi da sinistra verso destra: si posiziona lo pseudostato iniziale in alto a sinistra e lo stato finale in basso a destra.

Nel realizzare i diagrammi degli stati, molto importante è sindacare la presenza di stati destinatari di transizioni che non generano transizioni uscenti, e, caso opposto, stati da cui escono transizioni ma che non vi giungono. Chiaramente questi casi limite sono assolutamente validi per pseudostati iniziali e stati finali.

Nel caso in cui un diagramma degli stati risulti particolarmente complesso, è consigliato eseguire un'attività di revisione volta a verificare se sia possibile ridurne la complessità. Un meccanismo utilizzabile a tal fine consiste nel ricorrere a stati e/o transizioni composte. In particolare è consigliabile considerare l'introduzione di stati composti, raggruppanti diversi altri stati, nei casi in cui si evidenzi il caso in cui diversi stati prevedano transizioni, magari innescate da uno stesso evento, dirette verso gli stessi stati destinazione, oppure, caso contrario, siano destinatari di transizioni relative agli stessi eventi. Non è infrequente il caso in cui diversi diagrammi degli stati si inseriscano svariati stati fittizi, inclusi solo per necessità di modellazione. Talune volte risulta una situazione obbligatoria, mentre in altri casi si tratta esclusivamente di un errato utilizzo delle transizioni (specie di quelle interne), oppure del mancato utilizzo di uno stato storico. Un altro modo per agevolare la fruizione dei diagrammi degli stati consiste nell'organizzarli in modo gerarchico, favorendone una fruizione top-down. A tal fine si rivela particolarmente utile il ricorso a rappresentazioni collassate degli stati composti, l'utilizzo di stati stub e il ricorso a stati di sottomacchina. Molto importante è poi porre bene attenzione alle transizioni interne. Alcune volte può capitare di analizzare modelli aventi stati con azioni associate a transizioni di entrata e di uscita, per poi rendersi conto che le azioni devono essere eseguite solo in casi particolari. Ciò può essere risolto utilizzando opportunamente le condizioni di guardia, oppure rivedendo le transizioni di entrata e di uscita.

Qualora, nell'ambito di uno stato, uno stesso evento possa innescare diverse transizioni, la notazione delle macchine a stati prevede che ne sia scatenata solo una, la cui selezione avviene con criterio casuale. È opportuno analizzare per bene macchine a stati di questo tipo.

La notazione dei diagrammi di attività rappresenta una variante dei diagrammi degli stati, in cui le proprietà dell'elemento stato sono ridotte. Nel contesto dei diagrammi delle attività, la quasi totalità degli stati utilizzati rappresenta l'esecuzione di attività e pertanto, la maggior parte delle transizioni sono innescate implicitamente dall'evento di completamento delle predette attività. Le condizioni soddisfatte dall'oggetto durante la permanenza nello stato sono, invece, inerenti l'esecuzione delle attività stesse.

I diagrammi delle attività, analogamente a quelli di stato, sono associati a un classificatore, come un caso d'uso, un package, l'implementazione di un'operazione, ecc. e permettono di modellarne gli aspetti dinamici. In questo caso l'attenzione è focalizzata sui processi computazionali.

La quasi totalità degli stati presenti nei diagrammi di attività sono versioni semplificate denominati stati di azione. Queste sono caratterizzate dal disporre di un'azione associata all'evento di entrata e con, almeno, una transizione di uscita innescata implicitamente dalla terminazione della suddetta attività. Tipicamente a questi stati non è assegnato alcun nome e pertanto rappresentano istanze di stati anonimi. Graficamente, gli stati azione sono mostrati con una notazione molto simile a quella degli stati tradizionali: rettangolo dagli angoli smussati, con all'interno visualizzata l'espressione di azione (`action-expression`). La notazione utilizzabile per specificare tale espressione è totalmente demandata al modellatore: l'unico vincolo è che deve essere univoca all'interno dello stesso diagramma.

Un'ulteriore discendente dell'elemento stato, specializzazione dello stato di azione, è quello denominato stato di chiamata. Questo è caratterizzato dal disporre di un'unica azione, associata all'evento di entrata nello stato, che si risolve in un'invocazione di un metodo di un particolare oggetto. La notazione grafica è del tutto equivalente a quella degli stati di attività con una sola variante: nel rettangolo smussato è riportato sia il nome dell'operazione, sia quello del classificatore che la implementa, racchiuso da parentesi tonde.

Analogamente a quanto visto per gli stati di sottomacchina, la notazione dei diagrammi di attività prevede gli stati di sottoattività. Questi stati sono utilizzati per invocare stati di attività definiti in opportuni diagrammi specificati in altre parti del modello. Pertanto, rappresenta l'esecuzione di un insieme di passi non atomici che possono avere una certa durata: non è infrequente il caso di avere dei processi di sincronizzazione. Da un punto di vista pratico, questo stato può essere immaginato come un'invocazione di macro: ogni istanza di uno stato di sottoattività può essere sostituita dall'intero diagramma degli stati invocato. La notazione grafica degli stati di sottoattività prevede il medesimo rettangolo con gli angoli smussati, con l'aggiunta di un'apposita icona (due stati connessi attraverso una freccia) visualizzata nell'angolo in basso a destra.

Nei diagrammi di attività, la diramazione non concorrente del flusso è resa possibile attraverso le condizioni di guardia e gli pseudostati. In particolare, organizzando percorsi composti attraverso opportuni segmenti connessi da pseudostati, le condizioni di guardia permettono di selezionare i segmenti che compongono un determinato percorso tra quelli in alternativa; in altre parole, gli pseudostati e le condizioni di guardia permettono di suddividere il flusso del controllo. Nel contesto dei diagrammi di attività, gli elementi decisionali sono visualizzati attraverso la notazione classica del rombo.

Un diagramma di attività può essere suddiviso in un insieme di partizioni, ognuna delle quali è costituita da una collezione di elementi non condivisi. Queste partizioni sono denominate swimlane (corsie di nuoto) e permettono di organizzare i diagrammi delle attività secondo precisi criteri. Tipicamente, le swimlane sono associate a specifiche unità organizzative (come per esempio il sistema, sue parti, attori, particolari ruoli, ecc.) e pertanto permettono di enfatizzare l'allocazione delle attività, o, se si preferisce, l'assegnazione delle respon-

sabilità, alle diverse unità. Graficamente sono visualizzate attraverso rettangoli (con riportati all'interno gli stati appartenenti) dotati di un nome, visualizzato in alto.

La notazione dei diagrammi delle attività prevede la possibilità di visualizzare esplicitamente oggetti che sono input e/o output delle varie azioni. L'elemento utilizzato a tale fine è la relazione di flusso di oggetto, che definisce un flusso tra le azioni presenti nei diagrammi di attività. In particolare, sancisce che un'istanza di un particolare classificatore, tipicamente in un determinato stato, è disponibile quanto si entra nel relativo flusso di stato. La relazione di flusso di oggetto è visualizzato graficamente attraverso una freccia tratteggiata congiungente lo stato d'azione e il relativo oggetto. A seconda dell'utilizzo dell'oggetto, input o output, la direzione della freccia va, rispettivamente, dall'oggetto al relativo stato di attività o viceversa. Tipicamente, lo stesso oggetto è output di una determinata azione e input di una o più azioni successive. Qualora un oggetto sia visualizzato come output di un'azione, la generazione dell'oggetto è innescata dal completamento dell'esecuzione dell'azione. Quando uno stesso oggetto è utilizzato da diverse azioni, tipicamente, queste si occupano di variarne lo stato.

I diagrammi di attività definiscono una serie di stereotipi, tra i quali molto utili sono quelli utilizzati per evidenziare la ricezione e l'invio di segnali. Si tratta di stati sprovvisti di transizioni interne e di azioni. Per quanto concerne l'elemento relativo alla ricezione di un segnale, questo è visualizzato attraverso un pentagono concavo, con la forma di rettangolo con un triangolo entrante da uno dei suoi lati minori, con visualizzata internamente la firma del segnale. Volendo è possibile collegarvi lo stato che emette il segnale mediante una freccia tratteggiata. Per quanto riguarda l'elemento relativo all'invio di segnali, questo è mostrato graficamente attraverso un pentagono convesso con una forma speculare all'elemento precedente: un rettangolo con un triangolo uscente da uno dei suoi lati minori. Come nel caso precedente la firma è riportata all'interno del simbolo. Anche in questo caso è possibile associare lo stato con l'oggetto destinatario del segnale, mediante una freccia tratteggiata.

La notazione dei diagrammi di attività prevede la possibilità di gestire eventi differiti. Ciò permette di rimandare la gestione di determinati eventi che scaturiscono mentre una o più attività sono in corso di esecuzione. In particolare, ogni stato può dichiarare un insieme di eventi la cui gestione è differita qualora intervengano durante la permanenza nello stato stesso. Chiaramente l'insorgenza di questi eventi non innesca alcuna transizione. Qualora una transizione dipende da un particolare evento e questo sia presenta nella coda, la transizione è innescata immediatamente nel caso in cui non intervengono altri fattori a impedirlo. Qualora ci si trovi nella condizione in cui diverse transizioni potrebbero avere luogo, viene selezionata quella il cui evento si trova nella posizione più avanzata nella coda: l'evento più vecchio. La notazione utilizzata per mostrare gli eventi differiti, consiste nel menzionarli in un'apposita lista, con posposti la barra obliqua (`/`) e la parola chiave `deferred`.

Nei diagrammi di attività non è necessario mostrare esplicitamente stati di sincronizzazione. Pertanto è possibile collegare le transizioni concorrenti direttamente a partire dalle barre di sincronizzazione. L'utilizzo degli stati di sincronizzazione è invece necessario qualora, in condizioni di esecuzioni concorrenti, si voglia limitare la differenza tra il numero di transizioni entranti e quello delle transizioni uscenti da una barra di sincronizzazione. In questi casi la notazione rimane perfettamente equivalente a quella delle macchine a stati. I diagrammi di attività permettono di associare condizioni di guardia alle transizioni uscenti da una barra di sincronizzazione di tipo *fork*.

Le azioni specificate in uno stato d'azione o nel grafo di attività presente in uno stato di sottoattività possono essere eseguite diverse volte in maniera concorrente. Il numero delle invocazioni concorrenti è stabilito a tempo di esecuzione da un'espressione di concorrenza, la quale prevede come dominio una lista di argomenti, in cui ogni elemento è relativo a una singola invocazione. Per quanto concerne la notazione, qualora la concorrenza dinamica di un'azione assuma un valore diverso dall'unità, la relativa molteplicità è mostrata nell'angolo in alto a destra, altrimenti viene omessa.

Per quanto riguarda l'impiego dei daigrammi di attività, la discendenza diretta dei diagrammi di attività dai famosissimi diagrammi di flusso, nonché dai DFD (*Data Flow Diagram*) ne rende l'utilizzo quasi universale… I diagrammi delle attività si prestano a essere impiegati per miriadi di funzioni: analisi dei requisiti utente, modellazione dei workflow previsti dai processi di sviluppo del software, illustrazione di particolari meccanismi/procedure, disegno di algoritmi, specie paralleli, ecc.

Durante l'analisi dei requisiti si configurano come uno strumento validissimo per la cattura dei requisiti. In particolare, permettono di instaurare una magnifica piattaforma di colloquio con gli utenti, non senza effetti collaterali. In questo contesto possono essere utilizzati sia per descrivere gli scenari di un solo caso d'uso, sia per descrivere la collaborazione tra diversi. In quest'ultimo caso sono impiegati per modellare processi di business. Sempre in questa fase si prestano a descrivere formalmente business rule particolarmente complesse. Nel corso della fase di analisi, l'utilizzo è piuttosto limitato, per poi riprendere quota durante la fase di disegno. In questo ambito sono utilizzati per comunicare formalmente specifici algoritmi, specie se concorrenti. La notazione dei diagrammi di attività sembrerebbe particolarmente vantaggiosa per questo tipo di utilizzo.

Poiché la notazione dei diagrammi di attività rappresenta una particolare variante dei diagrammi di stato, molte delle linee guida di stile illustrate per i diagrammi di stato mantengono la loro validità anche nel contesto dei diagrammi di attività. In particolare, è necessario: sindacare l'assenza di pseudostati iniziali e degli stati finali, curare la posizione degli stessi pseudostati iniziali e degli stati finali e controllare i buchi neri e gli "stati dei miracoli".

Nei diagrammi di attività i punti decisionali sono rappresentati attraverso il simbolo del rombo. Però, a differenza delle notazioni passate, quali per esempio quella dei Flow Chart, non è possibile includervi la condizione che dà luogo alla diramazione. Quindi, ogni punto di decisione deve prevedere una transizione proveniente da un apposito stato nel quale si esegue l'azione i cui risultati possono generare la diramazione e assicurarsi che ogni transizione uscente da un punto di decisione sia corredata da apposita condizione di guardia. Inoltre è utile sia investigare l'utilizzo della parola chiave `else`, che spesso permette di raggruppare diverse transizioni, sia verificare che le transizioni uscenti da uno stesso punto di diramazione non siano dotate di condizioni di guardia che diano luogo a sovrapposizioni.

Qualora si decida di visualizzare attività concorrenti nei diagrammi di attività è necessario utilizzare esclusivamente le barre di sincronizzazione. Risulta importante assicurarsi che a ogni diramazione del flusso corrisponda una ricongiunzione, e che non siano presenti flussi generati da una diramazione che non confluiscano in alcuna barra di ricongiunzione.

Qualora si utilizzino delle swimlane, è opportuno disporre contiguamente swimlane in cui l'interazione è significativa e inoltre è consigliabile disporle in modo tale da enfatizzare la struttura a strati dell'architettura. Molto importante è anche limitare il numero di swimlane. Alcuni autori consigliano spesso di organizzare i

diagrammi di attività come se fossero presenti delle swimlane, ma senza visualizzarle esplicitamente. Ciò, molto frequentemente, permette di utilizzare in maniera più efficiente lo spazio a disposizione e quindi di produrre diagrammi più chiari.

La notazione dei diagrammi di attività consente di rappresentare esplicitamente gli oggetti utilizzati (input e/o output) da determinate operazioni. Chiaramente è inutile rappresentare esplicitante tutti gli oggetti manipolati all'interno di uno stesso diagramma: è sufficienti limitare l'attenzione a quelli che possono concorrere a aumentare il grado di comprensione del diagramma. Una seconda constatazione è relativa alla visualizzazione di tali oggetti. Nella quasi totalità dei casi, visualizzare una sola volta uno stesso oggetto, con molte linee entranti e uscenti, rende il diagramma confuso, e finisce per nascondere importanti flussi. Pertanto è consigliabile riportare uno stesso oggetto più volte, evidenziando, di volta in volta, lo stato in cui l'azione a cui è associato come output lo ha fatto transitare.

Un ultimo consiglio consiste nell'organizzare gerarchicamente complessi diagrammi di attività secondo dettami tipici di un approccio top-down. Ciò permette di focalizzare, in ogni istante, l'attenzione sui dettagli importanti per quel momento, rimandando lo studio degli altri dettagli. I diagrammi di attività favoriscono tale approccio grazie alla presenza degli stati di sottoattività.

Con il presente capitolo si conclude l'illustrazione delle notazioni fornite dallo UML per modellare gli aspetti dinamici del sistema.