Modellazione discreta per attività, esempi

Simulazione & Logistica, I modulo
Esercitazione n. 5

Corso di Laurea in Informatica Applicata
Università di Pisa, sede di La Spezia
A.a. 2008/09, I semestre

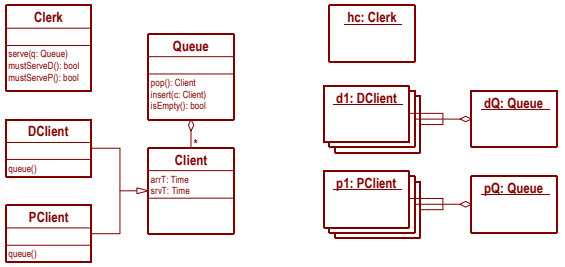
Contenuti

- L'impiegato sempre tormentato (ad attività)
- Modellazione e simulazione per attività
  - Strutture dati a tempo di esecuzione
  - Specializzazione delle classi attività
  - Frammenti di codice
  - Corrispondenza con il modello
  - Conversione a 3 fasi
- Considerazioni, problemi e suggerimenti
  - Dipendenza dalle tecnologie
  - Gestione degli eventi esterni

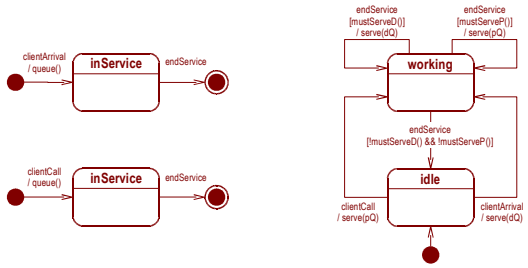
L'impiegato tormentato (ad attività)

- Un servizio al pubblico
  - I clienti arrivano di persona
  - Oppure telefonano
  - L'impiegato serve i clienti allo sportello o al telefono
  - I clienti che arrivano allo sportello aspettano (in ordine)
  - Il centralino ordina le telefonate in attesa
  - I clienti allo sportello hanno la priorità
  - I tempi di servizio dipendono dai clienti
- Modellazione ad attività
  - Classica e con il metodo delle 3 fasi
  - Interpretazione "libera" del diagramma degli stati

Classi e oggetti



Macchine a stati



Identificare le attività

- Identificare gli stati “attivi”, quindi le attività
  - Per entrare in uno stato attivo
  - Per concluderlo
- Una classe per ogni elemento del vettore
- Attività dell’impiegato
  - ACkSvD, ACkSvP, inizio servizio allo sportello e al telefono
  - ACkEnd, fine del servizio
- Attività dei clienti allo sportello e al telefono
  - ADCArv, APCCll, arrivo del cliente
  - ADCEnd, APCEnd, fine del servizio per quel cliente
- Gli stati, comunque, conviene non dimenticarli

Specificare le attività

- Definire tempi e guardie
- Il diagramma va interpretato
  - Le attività sugli archi non sono le azioni
  - Tempi e guardie si ricavano dopo aver identificato le attività
- Attività dipendenti da condizioni
  - ACkSvD, ACkSvP: se ci sono clienti in coda
- Attività dipendenti dal tempo
  - ACkEnd: quando la richiesta è servita
  - ADCArr, APCCll: quando arriva un cliente
  - ADCEnd, APCEnd: quando la richiesta del cliente è servita

Vettore delle attività

- Il vettore di attività

attività	tempo	checkGuard()	doAction()
ADCArv	clientArrival	-	client.queue()
APCCll	clientCall	-	client.queue()
ACkEnd	endService	!hc.idle	hc.idle = true
ADCEnd	endService	client != NULL	delete client
APCEnd	endService	client != NULL	delete client
ACkSvD	-	hc.idle && mustServeD()	serve desk
ACkSvP	-	hc.idle && mustServeC()	serve phone

- Note
  - Notazione: `ActArray::a[adcarv] = new ADCArv()`
  - Le attività ADC* e APC* riferiscono un cliente
  - Gli stati devono essere gestiti esplicitamente
  - Conviene che l'impiegato vada sempre idle (3 fasi ...)
  - Servire significa (ri)definire il tempo delle A*End

Classi e metodi, rivisti

- Classi per le attività
  - Metodi doAction() specializzati per ogni attività
  - Specializzazione delle classi ADC* e APC* aggiungendo un attributo per il cliente corrente
- Impiegato: metodi serve esplicitati nelle attività
  - ```
    void ACkSvD::doAction() {
      Client c = dQ.pop();
      cal.setTime(ackend, curT + c.srvT);
      cal.a[adcend].client = c;
      cal.setTime(adcend, curT + c.srvT);
      hc.idle = false;
    }
    ```
- Da eventi ad attività
  - Programmare un evento può diventare le ridefinizione del tempo di più di una attività
    ```

Un problema

- Attività corrispondenti a eventi esterni
- L'agenda è una struttura a dimensione fissa
  - Non si possono caricare prima tutte le istanze delle attività
  - Chi (ri)definisce il tempo di ADCArv e APCCll?
  - Chi crea i clienti?
- Soluzione obbligata: le attività si ridefiniscono
- Arrivo dei clienti allo sportello

```
void ADCArv::doAction() {
  dQ.insert(client);
  client = new Client(genArrT(), genSrvT());
  t = client.arrT;    // cal.setTime(adcarv, c.arrT);
}
```


Attività in esecuzione

- Aggiornare il tempo
  - Determinato da ADCArv, APCCll, A*End
  - Non si considerano le attività indipendenti dal tempo
- Effetti dell'esecuzione
  - ADCArv e APCCll si redifiniscono il tempo e si disabilitano, modificando le code abilitano ACkSvD e ACkSvP
  - ACkEnd aggiorna lo stato dell'impiegato disabilitandosi, eventualmente abilitando ACkSvD e ACkSvP
  - ADCEnd e APCEnd aggiornano lo stato, disabilitandosi
  - ACkSvD e ACkSvP aggiornano il tempo ad ACkEnd e ad ADCEnd e APCEnd disabilitandole tutte, aggiornano lo stato dell'impiegato disabilitandosi
- Occorre tener conto delle passate e dell'ordinamento

Considerazioni

- Il metodo a tre fasi è abbastanza naturale
  - Nell'esempio le attività sono le stesse
  - Anzi, si eliminano delle guardie introdotte artificialmente
- Non sempre si disaccoppia il codice
  - Non basta sapere che il cliente è interessato alla fine del servizio (doAction() e handle())
  - Le attività ACkSvD e ACkSvP devono conoscere anche le attività "del cliente" il cui tempo deve essere ridefinito
- Il diagramma UML è distante dal codice
- Generare gli eventi può essere una buona idea