Modellazione discreta con UML, esempi

Simulazione & Logistica, I modulo
Esercitazione n. 3

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

Contenuti

- L'impiegato tormentato con UML
  - Confronto fra notazioni: cicli di attività vs UML
  - Caratteristiche di UML per l'analisi e la specifica del sistema
- Uso di UML
  - Gentile, traduzione intuitiva dei cicli di attività
  - Tosto, variazioni usando lo stile e l'espressività di UML
  - Diverse possibilità offerte dal linguaggio
- Verso l'implementazione
  - Obiettivo di una convenzione d'uso di UML
  - Corrispondenza fra diagrammi e codice

L'impiegato tormentato (con UML)

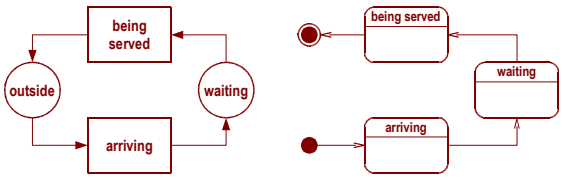
- Un servizio al pubblico
  - I clienti arrivano di persona
  - Oppure telefonano
  - L'impiegato serve i clienti allo sportello o al telefono
  - I clienti che arrivano allo sportello aspettano (in ordine)
  - Il centralino ordina le telefonate in attesa
  - I clienti allo sportello hanno la priorità
  - I tempi di servizio dipendono dai clienti
- Modellazione in UML
  - Come sintassi alternativa a quella classica dei cicli di attività
  - Come ampliamento e raffinamento dei cicli di attività

Analisi, dove eravamo arrivati

- Entità
  - Classi: impiegato, cliente allo sportello, cliente al telefono
  - Oggetti: 1 impiegato, *h*+*k* clienti, allo sportello e al telefono
  - Insiemi: coda allo sportello, coda gestita dal centralino
- Operazioni
  - Eventi: arrivo di un cliente allo sportello *esterno*
    arrivo di una telefonata *esterno*
    fine del servizio (sportello o telefono) *interno*
  - Attività: il cliente arriva e si mette in coda
    il cliente telefona e il centralino lo mette in attesa
    il cliente aspetta (allo sportello o al telefono)
    l'impiegato serve (allo sportello o al telefono)
    il cliente se ne va (fisicamente o riattaccando)
    l'impiegato aspetta (ozia ...)

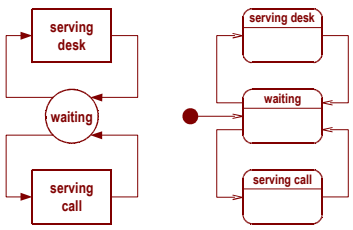
Uso gentile - il cliente

- Quadrare i tondi, arrotondare i quadrati
  - Cambiare la notazione, mantenendo la corrispondenza ...
    ... ma aggiungendo poca informazione alla specifica
  - Cliente allo sportello (arriving) e al telefono (calling)



Uso gentile - l'impiegato

- Identificazione della condizione iniziale
  - Aggiunta per rispetto della sintassi UML
  - Non è un'informazione inutile

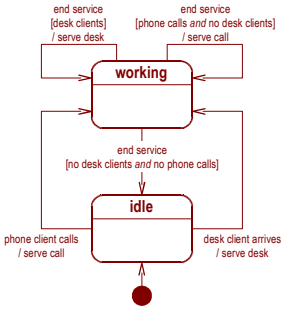


"Stile" UML

- Eventi
  - Gli eventi sono fatti, comportano azioni
  - Gli stati sono condizioni di attesa di particolari eventi
  - Sintassi: eventi, condizioni e azioni sulle transizioni
- Attività
  - Un oggetto si attiva in risposta a un evento
  - Attività interne in/ e out/ come fattorizzazione
  - Attività interne do/ solo se continue (nessuna!)
- Tecniche di modellazione con UML
  - Concentrarsi sugli aspetti "attivi" delle entità
  - Conservare gli stati in cui si fa qualcosa o si attende
  - Fattorizzare

Uso quasi tosto - l'impiegato

- Eventi, condizioni e azioni: notazione informale
- Eventi
  - Definiscono "quando" la transizione può scattare
- Condizioni
  - Definiscono "se" la transizione può scattare
- Azioni
  - Definiscono "cosa" succede
  - Sono istantanee

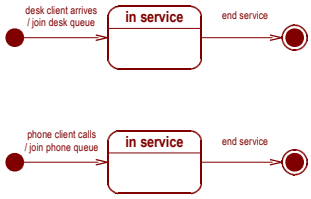


L'impiegato: note e riflessioni

- Uso di UML
  - Non è una specifica, ancora modo bozza o disegno
  - Aggiunge: eventi, condizioni
  - Stati come attesa di eventi
  - Eventi come transizioni (con condizioni e azioni)
- Tecniche di modellazione: fattorizzazione
  - Degli stati: l'impiegato lavora o è in attesa
  - Degli eventi: arrivi distinti (priorità), fine del servizio unica
- Cosa manca ancora:
  - Come si verificano le condizioni?
  - Come si specificano le azioni?
  - Come si trattano gli insiemi (le code)?
  - Come avvengono gli eventi?

Uso quasi tosto – i clienti

- Fattorizzazione
  - Transizioni: eliminate quelle senza azioni (a volte dipendenti da eventi di altri clienti)

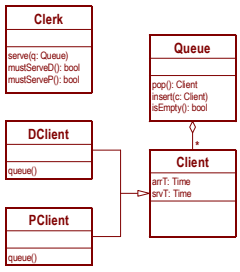


Specificare condizioni e azioni

- Oggetti con cui specificare condizioni e azioni
  - Attributi, per esprimere condizioni
  - Metodi, per definire azioni
  - Classi per definirle, oggetti per renderle concrete
- Condizioni presenti nel problema
  - Ci sono clienti in attesa allo sportello
  - Ci sono clienti in attesa al centralino
  - Sono condizioni su code
  - La presenza di code era identificata già dai cicli attività
- Azioni (metodi con parametri)
  - Servire un cliente (toglierlo dalla giusta coda)
  - Stabilire la durata di un servizio (dipende dal cliente)

Uso tosto – classi

- Diagramma delle classi
  - Attributi (cliente):
    - tempo di arrivo
    - tempo di servizio
  - Metodi (clienti, per tipo):
    - per mettersi nella propria coda
  - Metodi (impiegato):
    - esecuzione del servizio
    - predicati per la politica
- La coda
  - Esplicitata, ma ragionevolmente è un componente di libreria



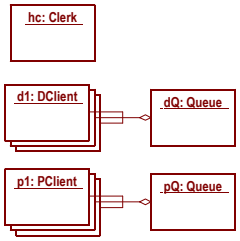
Uso tosto - oggetti

- Diagramma degli oggetti
  - Un impiegato, $h+k$ clienti
  - Code e partizione dei clienti
- Si possono scrivere i metodi

```
void DClient::queue() {
  dQ.insert(this);
}
```

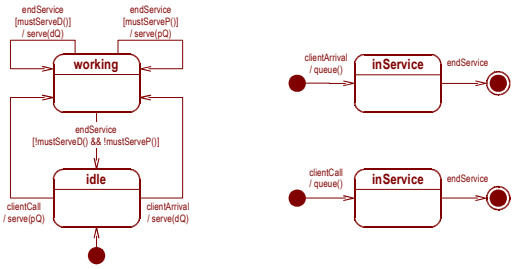
```
bool Clerk::mustServeD() {
  return !dQ.isEmpty();
}
```

```
void Clerk::serve(Queue q) {
  Client c = q.pop();
  schedule(endService,
           curT() + c.srvT);
}
```



Uso tosto - macchine a stati

- Per le classi Clerk, DClient e PClient



Verso l'implementazione

- Pattern di traduzione
  - Gestione di un evento
  - Ogni classe attiva ha il suo
  - Dalla macchina a stati, ...
  - ... codificando stati ed eventi
  - Switch/if-else nidificati:
    - stati (enumerati, 1 .. *i* .. *s*)
    - archi uscenti (1 .. *n*)
  - Per ogni arco uscente:
    - evento (enumerati)
    - guardia (condizione)
    - azione (metodi)

```
void ActiveEntity::handle(Event e) {
  switch (curState) {
    ...
    case Si:
      if (e == Ei1 && Gi1) {
        Ai1();
        curState = Si1;
      } else
      ...
      if (e == Ein && Gin) {
        Ain();
        curState = Sin;
      }
      break;
    ...
  }
}
```

Esempi

- Per le classi
  - DClient
  - Clerk

```
void DClient::handle(Event e) {
  switch (curState) {
    case initial:
      if (e == clientArrival) {
        queue();
        curState = inService;
      } break;
    case inService:
      if (e == endService) {
        delete this;
      } break;
  }
}
```

```
void Clerk::handle(Event e) {
  switch (curState) {
    case idle:
      if (e == clientArrival) {
        serve(dQ);
        curState = working;
      } else
      if (e == clientCall) {
        serve(pQ);
        curState = working;
      } break;
    case working:
      if (e == endService && mustServeD()) {
        serve(dQ);
      } else
      if (e == endService && mustServeP()) {
        serve(pQ);
      } else
      if (e == endService &&
          !mustServeD() && !mustServeP()) {
        curState = idle;
      } break;
  }
}
```


Assunzioni e domande

- Trucchetto: si passa sempre dalle code
  - I clienti arrivano (telefonano) e si mettono sempre in coda, l'impiegato preleva dalle code certo di trovarceli
  - Implicazione: Client::handle() prima di Clerk::handle()
- Priorità fra eventi
  - Cosa succede se un cliente arriva insieme a una chiamata?
  - Implicazione: occorre un ulteriore ordinamento (priorità)
- Gestione del tempo e degli eventi
  - Sappiamo che il tempo scorre e che si può conoscere
  - Sappiamo che si possono programmare gli eventi interni
  - Chi tiene conto del tempo?
  - Chi fa scattare gli eventi e invoca i metodi handle()?
  - E gli eventi esterni?