

Sim
UniPisa
LaSpezia

L'impiegato tormentato: variazioni sul tema (in UML)


Simulazione – Esercitazione n. 3
Corso di Laurea in Informatica Applicata
Università di Pisa, sede di La Spezia

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 1/18 

Sim
UniPisa
LaSpezia

Contenuti


- Hardboiled OO
- Politiche sofisticate e obiettivi
- Clienti impazienti
- Necessità dell'impiegato
- Generazione di eventi esterni

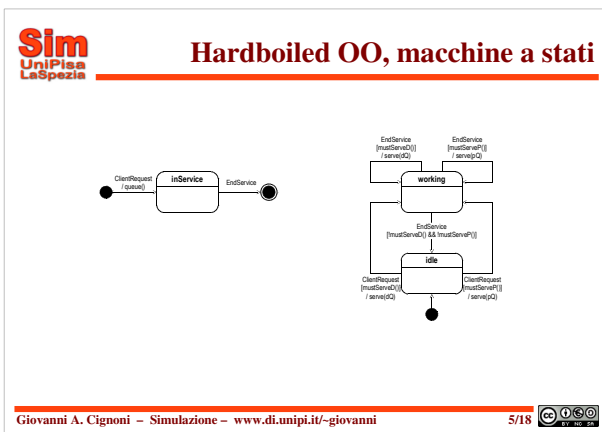
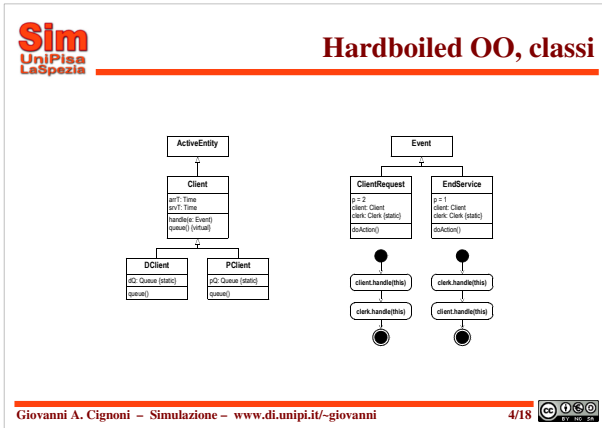
Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 2/18 

Sim
UniPisa
LaSpezia

Hardboiled OO

- Formulazione originale
- Tentazioni dal modello UML
 - Le macchine a stati dei clienti sono molto simili (e semplici)
 - Esiste già una classe Client generica
 - Fattorizzare il metodo handle()
- Conseguenze
 - Necessità di un solo evento di "ingresso"
 - Fattorizzare ClientArrival e ClientCall in ClientRequest
 - L'impiegato reagisce a ClientRequest e decide con i predicati
- Funziona, è più compatto, forse è eccessivo

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 3/18 



- Sim UniPisa LaSpezia** **Politiche sofisticate e obiettivi**
- **Politica parametrica**
 - La priorità è ai clienti allo sportello, finché la coda al centralino non supera di una soglia k la coda allo sportello
 - Politica originale: $k = \infty$ (o molto grande)
 - **Obiettivi di simulazione**
 - Tener traccia del tempo di ozio dell'impiegato
 - **Modifiche, locali all'impiegato**
 - La soglia k , contatori e metodi per il conteggio del tempo idle
 - I predicati `mustServeD()` e `mustServeP()`
 - Il metodo `handle()` per aggiornare il tempo di ozio totale
 - Eventi e clienti riusati
- Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 6/18

Sim
UniPisa
LaSpezia

Politiche sofisticate, diagrammi

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 7/18

Sim
UniPisa
LaSpezia

Politiche sofisticate, metodi

```

bool Clerk::mustServeD() {
    if (dQ.isEmpty()) return false;
    else if (pQ.isEmpty()) return true;
    else return (pQ.length() - dQ.length() <= plcyTh);
}

bool Clerk::mustServeP() {
    if (pQ.isEmpty()) return false;
    else if (dQ.isEmpty()) return true;
    else return (pQ.length() - dQ.length() > plcyTh);
}

Clerk::startIdle() { idleS = curT; }

Clerk::stopIdle() { idleT += curT - idleS; }
    
```

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 8/18

Sim
UniPisa
LaSpezia

Clienti impazienti

- Le code sono infinite, la pazienza dei clienti no
 - Ogni cliente ha un tempo limite di attesa
 - I clienti al telefono scaduto il limite riattaccano
 - I clienti allo sportello valutano in base alla posizione in coda
- Obiettivi della simulazione
 - Registrare i clienti persi e quelli serviti in ritardo
- Modifiche
 - Un evento (interno) in più: la “sveglia” del cliente
 - Nuovi attributi per il cliente: attesa e posizione accettabile
 - Metodo handle(): uno stato apposta per gli eventi pendenti
 - Eventi già definiti e impiegato riusati

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 9/18

Sim
UniPisa
LaSpezia

Clienti impazienti, diagrammi

The diagram shows the following classes and their relationships:

- Event**: Base class for ClientBored and Client.
- ClientBored**: Inherits from Event. Attributes: p: int, client: Client, (lock: any). Method: clientHandled().
- Client**: Inherits from Event. Attributes: wait: Time, curT: Time, pos: Time, queue: (virtual). Method: queue().
- DClient**: Inherits from Client. Attributes: dQ: Queue (static), gPos: Integer. Methods: queue(), decideToLeave(), leave(), handle(e: Event).
- PClient**: Inherits from Client. Attributes: pQ: Queue (static). Methods: queue(), decideToHangUp(), handle(e: Event).
- inService**: State of a client being served. Attributes: ClientBored (handleToLeave()), ClientBored (handleToLeave()).
- late**: State of a client being late. Attributes: ClientBored (handleToLeave()).
- served**: State of a client being served.

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 10/18

Sim
UniPisa
LaSpezia

Clienti impazienti, metodi

```

DClient::queue() {
    dQ.pushLast(this);
    schedule(new ClientBored(curT + wait));
}
bool DClient::decideToLeave() {
    if (dQ.getPos(this) <= gPos) // -1 out, i.e. served
        return false;
    else return true;
}
bool PClient::decideToHangUp() {
    if (dQ.isIn(this)) return true;
    else return false;
}
DClient::leave() { dQ.extract(this) }
    
```

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 11/18

Sim
UniPisa
LaSpezia

Clienti impazienti: II implementazione

- Evitare di dover aspettare eventi ormai inutili
 - Possibilità di ritirare gli eventi noti
 - Opportunità di modellazione offerta dal particolare motore
 - `ActiveEntity::withdraw(GS_EvUId eUId)`
- Dilemma temporale: è lecito modificare il futuro?
 - A certe condizioni
 - Il futuro è noto ed è proprio
 - Un'entità può eliminare gli eventi da lei programmati
- Diagramma della macchina a stati semplificato

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 12/18

Sim UniPisa LaSpezia **Clienti impazienti 2, dettagli**

```

DCClient::queue() {
    dQ.pushLast(this);
    Event e = new ClientBored(curT+wait);
    bEv = e.getUId();
    schedule(e);
}
    
```

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 13/18


Sim UniPisa LaSpezia **Necessità dell'impiegato**

- Ogni tanto l'impiegato ha necessità impellenti
 - Se non ci sono clienti, approfitta immediatamente
 - Altrimenti, per un certo tempo, aspetta la prima pausa utile
 - Stremato, finito il servizio corrente, si concede una pausa
- Interpretazione di idle (studiato dalla simulazione)
 - Tempo utile per altre attività (paused != idle)
- Modifiche, locali all'impiegato
 - Tre eventi in più: necessità, fine dell'attesa e fine della pausa
 - Nuovi attributi e metodi per l'impiegato
 - Metodo handle(): tre stati in più
 - Eventi già definiti e clienti (impazienti) riusati

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 14/18

Sim UniPisa LaSpezia **Necessità dell'impiegato, diagrammi**


Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni 15/18



Necessità dell'impiegato, metodi


```


Clerk::hold() {
    schedule(new CantWait(curT + hldT));
}
Clerk::haveRelief() {
    schedule(new EndPause(curT + rlfT));
}
Clerk::nextNeed() {
    schedule(new Need(curT + nndT));
}
    
```



- Eventi da ritirare?

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni


16/18 




Generazione di eventi esterni

- Per esempio gli arrivi e le chiamate dei clienti
- Precaricamento, possibile ma inefficiente
- Generazione da parte delle entità del sistema
 - Tempi fra un evento e il successivo
 - Generazione dell'evento come parte della sua gestione
- Generazione da parte di entità specifiche
 - Soluzione standard che approfitta dell'architettura
 - Mantenere l'esternalità degli eventi rispetto al "vero" sistema
 - Favorire il disaccoppiamento

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni

17/18 



Generazione eventi esterni, dettagli

```

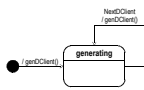
DCGen::genDClient() {
    time t = curT + dArv;
    dc = new Dclient(t, srvT, waiT, gPos);
    schedule(new ClientArrival(dc));
    schedule(new NextDClient(t));
}
    
```

ActiveEntity

↓

DCGen

gArv: Time (RVar)
 gArT: Time (RVar)
 gPos: Time (RVar)
 gPos: Range (RVar)
 genDClient()
 handle(Event)



Event

↓

NextDClient

t: Time
 gen: DCGen (static)
 doAction()

↓

gen.handle(t)

Giovanni A. Cignoni – Simulazione – www.di.unipi.it/~giovanni

18/18 