

## Programmazione di un Simulatore a Eventi Discreti

Michela Meo

Maurizio M. Munafò

Michela.Meo@polito.it - Maurizio.Munafò@polito.it

## Copyright

- Quest'opera è protetta dalla licenza Creative Commons NoDerivs-NonCommercial. Per vedere una copia di questa licenza, consultare: <http://creativecommons.org/licenses/nd-nc/1.0/> oppure inviare una lettera a: Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.
- This work is licensed under the Creative Commons NoDerivs-NonCommercial License. To view a copy of this license, visit: <http://creativecommons.org/licenses/nd-nc/1.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

## Scheduling degli eventi

- Consideriamo il problema della codifica di modelli basati sullo scheduling di eventi discreti con linguaggi general-purpose
- Occorre realizzare le funzioni di gestione degli eventi e di controllo della simulazione

## Simulatore ad eventi discreti

- Modello a scheduling degli eventi
- Realizziamo un **simulatore ad eventi discreti in linguaggio C**
- Occorre realizzare le funzioni di gestione degli eventi e di controllo della simulazione

## Struttura del simulatore

- Gli elementi costitutivi di un simulatore ad eventi discreti sono generalmente:
  - il ciclo principale (Event Loop) di analisi degli eventi
  - l'insieme di eventi futuri (Future-Event Set)
  - le procedure da eseguire alla realizzazione di ogni evento
- A questi elementi chiave si aggiungono:
  - le procedure di raccolta ed analisi delle grandezze che intendiamo misurare
  - le procedure di inizializzazione del simulatore
  - il criterio di terminazione della simulazione

## Strutture dati

- Prima di definire in dettaglio le diverse parti del simulatore, occorre definire che tipo di informazioni fondamentali dovremo rappresentare e come
  - Il tempo di simulazione
  - Gli eventi e i loro attributi

## Il tempo di simulazione

- Il tempo di simulazione è una rappresentazione compressa/dilatata del tempo nel sistema reale
- La nostra rappresentazione numerica del tempo può essere:
  - floating-point
  - intera

## Il tempo di simulazione

- La rappresentazione del tempo in formato **floating-point** è quella più vicina alla natura reale del tempo
- La probabilità che due eventi accadano esattamente nello stesso istante di tempo è nulla
- Possiamo quindi non preoccuparci in maniera particolare della gestione di eventi contemporanei

## Il tempo di simulazione

- La rappresentazione in formato **intero** è adeguata per quei sistemi in cui il tempo procede naturalmente a scatti (sistemi a slot)
- Diventa significativa la probabilità che più eventi accadano nel medesimo istante di tempo discreto
- Occorre quindi definire un criterio di ordine tra eventi contemporanei

## Il tempo di simulazione

- Indipendentemente dal formato di rappresentazione, ci serve un indicatore globale dello scorrere del tempo: l'orologio di sistema (*clock time*)
- È una variabile globale, accessibile da ogni punto del simulatore, che rappresenta il valore corrente del tempo di simulazione

## Eventi: struttura dati

- Ogni evento dovrà essere rappresentato nel nostro programma come una struttura dati composta, la cui natura dipenderà da come si è scelto di implementare il Future-Event Set
- Ogni evento conterrà almeno:
  - il tempo in cui l'evento deve accadere
  - un identificatore del tipo di evento

## Eventi: struttura dati

- Il tempo di scheduling dell'evento è il campo chiave, dal momento che ci serve per decidere qual è il prossimo evento da eseguire, ovvero quello con tempo di scheduling minore in tutto il FES
- Il tipo di evento ci serve ad identificare quale tipo di evento deve accadere, e quindi quale particolare funzione debba essere eseguita

## Eventi: struttura dati

- È spesso utile associare ad ogni evento degli attributi
- Questo ci permette di aggregare eventi in base alla loro tipologia generale, per poi discriminare in fase di esecuzione
- Esempio
  - In un sistema a coda con due diversi tipi di servitore, invece di definire eventi distinti per i due servitori, definiamo un unico evento di servizio e gli associamo come attributo il servitore coinvolto

## Event Loop

- Il cuore di un simulatore ad eventi discreti è l'Event Loop
- Si tratta di un ciclo, all'interno del quale:
  - si estrae dal FES il prossimo evento
  - si fa avanzare il *clock time* al tempo di scheduling dell'evento
  - in base al tipo di evento si esegue la procedura associata

## Event Loop: pseudo-codice

```
while (condizione di terminazione)
{
  preleva evento from Future-Event Set;
  current_time = evento->time;
  switch (evento->type)
  {
    case TYPE1: procedure_type1();
    break;
    case TYPE2: procedure_type2();
    break;
    ...
  }
  rilascia evento;
}
```

## Event Loop: terminazione

- L'Event Loop viene eseguito fino al raggiungimento della condizione di terminazione
- Alcune possibili condizioni
  - Non ci sono più eventi nel FES: il sistema si è naturalmente esaurito
  - Si è raggiunto un tempo massimo di simulazione prestabilito
  - Si è raggiunto un numero massimo di eventi prestabilito
  - Si sono raggiunte adeguate condizioni di precisione sulle misure effettuate

## Future-Event Set

- Il Future-Event Set è la struttura chiave di un simulatore ad eventi discreti
- Dalla sua implementazione dipende l'efficienza del simulatore
- Dovremo considerare il trade-off tra semplicità implementativa ed efficienza
  - le strutture più semplici sono spesso le meno efficienti mentre le più efficienti possono essere complesse da implementare

## Future-Event Set

- Il Future-Event Set è l'insieme di tutti gli eventi di cui è prevista l'esecuzione
- Da tale insieme, di volta in volta, preleveremo l'evento con il tempo di scheduling minimo (Next Event)
- Qualsiasi struttura dati può essere usata per rappresentare il FES, ma ovviamente alcune strutture sono più adatte e più efficienti di altre

## Future-Event Set

- La struttura dati impiegata deve supportare nella maniera più efficiente possibile le operazioni tipiche richieste sul FES
  - Inserimento di un nuovo evento
  - Estrazione del Next Event
  - Ricerca e cancellazione di un evento
- L'efficienza di gestione dei primi due eventi è fondamentale, mentre la gestione della ricerca e cancellazione può essere richiesta solo in casi abbastanza avanzati

## Future-Event Set

- L'efficienza di gestione dipende però molto dalle condizioni particolari della simulazione
  - La sequenza con cui sono effettuate le operazioni di inserimento e cancellazione
  - Il numero di eventi mediamente contenuto nella FES
- Ci dobbiamo accontentare di valutazioni dell'efficienza nel *worst-case* e del suo ordine di grandezza

## FES: Lista lineare

- Il modo più semplice per realizzare il FES è quello di usare una lista lineare
- Per ottimizzare le operazioni di inserimento, estrazione e cancellazione la lista può essere
  - ordinata secondo il tempo di scheduling degli eventi
  - bilincata

## FES: Lista lineare

- Inserimento (ordinato)
  - Può avvenire indifferentemente dalla testa o dalla coda della lista
  - Nell'ipotesi di schedulare tendenzialmente eventi remoti, l'inserimento dal fondo può essere preferibile
  - Per N eventi già presenti in lista, la complessità di inserimento è  $O(N)$

## FES: Lista lineare

- Estrazione
  - Essendo la lista ordinata, richiede solo la rimozione del primo evento nella lista
  - Complessità  $O(1)$
- Ricerca e cancellazione
  - La ricerca ha la stessa complessità di un inserimento,  $O(N)$
  - Se si sa di dover cancellare degli eventi, può essere utile mantenerne un riferimento esterno al FES, per rimuoverli senza doverli cercare → complessità  $O(1)$

## FES: Lista lineare

- La complessità di gestione è quindi in genere  $O(N)$
- Il valore di N dipende dalle condizioni di funzionamento del sistema
- Per esempio, spesso:
  - In condizioni di carico basso, N è piccolo
  - In condizioni di sovraccarico, che sono quelli che spesso ci interessano maggiormente, N può essere molto grande

## FES: Liste lineari multiple

- Possiamo migliorare sensibilmente l'efficienza di utilizzo delle liste lineari usando più liste lineari in parallelo, una per ogni tipo evento
- All'estrazione, preleviamo il minimo tra gli eventi in cima ad ogni lista
- L'inserimento è ora  $O(N_i)$  e l'estrazione  $O(m)$ , essendo  $N_i$  il numero di eventi di tipo  $i$  ed  $m$  il numero di tipi di eventi

## FES: Heap

- Una soluzione per ridurre la complessità di inserimento è quella di usare strutture diverse da una lista lineare
- Uno heap è un particolare tipo di struttura ad albero in cui gli elementi sono parzialmente ordinati e che viene generalmente mantenuto mappato su un array

## FES: Heap

- Lo heap binario ha complessità di inserimento e di estrazione  $O(\log_2 N)$ , legata alla necessità di riposizionare alcuni elementi dell'albero binario in coincidenza di tali eventi
- Oltre alla complessità di inserimento ed estrazione è necessario considerare l'*overhead* necessario per riallocare l'array sul quale lo heap è mappato al crescere del numero di elementi in esso contenuti

## FES: Strutture ibride

- È possibile definire strutture dati per il FES organizzate in maniera ibrida, in maniera tale da offrire la minor complessità possibile
- Un tipo di soluzione particolarmente efficiente è quella di mantenere gli eventi in una lista lineare bilincata ed aggiungervi una "sovrastuttura" di puntatori a posizioni intermedie in tale lista

## FES: Strutture ibride

- Una tale soluzione offre ancora complessità  $O(1)$  per l'estrazione, mentre l'inserimento ha complessità inferiore ad  $O(N)$ , dal momento che l'inserimento è facilitato dall'accesso diretto a posizioni intermedie del FES
- Occorre considerare l'aumento di complessità introdotto dalla gestione della sovrastuttura

## FES: Conclusioni

- La gestione del FES è cruciale per il funzionamento efficiente del simulatore
- Occorre considerare il *trade-off* tra la complessità di gestione del FES e il guadagno di efficienza e di tempo di esecuzione

## FES: Conclusioni

- FES a lista lineare
  - semplici da implementare
  - adeguati se il sistema non è complesso e il numero di eventi limitato
- FES ad heap o con strutture ibride
  - necessarie per sistemi in cui debba essere manipolato un elevato numero di eventi
  - prestazioni ridotte rispetto alle liste lineari nel caso di un numero di eventi limitato, a causa dell'overhead di gestione

## FES: Esempio

- Vediamo brevemente le caratteristiche del FES che useremo durante le nostre esercitazioni di laboratorio
- Si tratta di un FES basato su una lista bilincata circolare con inserimento dal fondo
- L'allocazione/rilascio degli eventi è gestita tramite un meccanismo di riciclo delle strutture dati (*free list*)

## FES: Esempio

- La lista è circolare (per semplicità di gestione): verrà referenziata all'interno del programma da un puntatore allocato allo scopo
  - Event \*event\_list
- Tale puntatore si riferisce all'ultimo elemento della lista

## FES: Esempio

- Inserimento di un evento
  - void insert\_event(Event \*\*last, Event \*elem)
  - last è il puntatore al FES, elem l'evento da inserire
- Estrazione di un evento
  - Event \*get\_event(Event \*\*last)
  - ritorna l'evento in cima al FES
- Creazione di un evento
  - Event \*new\_event()
  - ritorna un nuovo evento, prelevandolo dalla *free list* o allocandolo effettivamente in memoria

## FES: Esempio

- Distruzione di un evento
  - void release\_event(Event \*elem)
  - rilascia l'evento elem, inserendolo nella *free list* per riciclare la memoria; deve essere invocata dopo l'esecuzione dell'evento
- Il ciclo di vita di ogni evento è quindi:
  - new\_event -> insert\_event -> get\_event -> release\_event

## FES: Esempio

- Anche se non strettamente necessaria, ci può servire definire una funzione
  - void schedule(int type, Time time, ...)
  - che realizzi lo scheduling di un evento di tipo type al tempo time, ovvero
    - allochi un nuovo evento
    - ne inizializzi la struttura dati con i valori di tipo, tempo di esecuzione ed eventuali altri parametri che sia necessario memorizzare nell'evento
    - inserisca l'evento nel FES

## Event Callback

- Sono le funzioni che realizzano, sotto forma di programma, il diagramma di flusso con cui abbiamo modellato ogni nostro evento
- Sono eseguite nell'Event Loop alla realizzazione dell'evento corrispondente
- Al loro interno, come abbiamo visto dai nostri modelli, avviene lo scheduling di nuovi eventi e quindi l'evoluzione della simulazione

## Raccolta ed analisi delle misure

- Il nostro obiettivo è quello di valutare le prestazioni del sistema
- Gli indici di prestazione che ci interessano variano di caso in caso ma sono rappresentati da grandezze numeriche misurabili all'interno del programma
- Azione chiave della nostra simulazione è raccogliere campioni di tali grandezze numeriche per poi procedere alla valutazione dei nostri indici di prestazione

## Cosa vogliamo misurare?

- Grandezze puntuali
  - Numero di realizzazioni di un certo evento (es. per la coda: numero totale di arrivi, partenze, perdite, etc.)
  - Valore massimo o minimo di una certa grandezza (ritardo, lunghezza di una coda, numero di utenti presenti nel sistema, etc.)
- Per questo tipo di misure ci bastano degli accumulatori da incrementare o aggiornare durante l'esecuzione del programma

## Cosa vogliamo misurare?

- Grandezze mediate
  - Tempi medi (di permanenza nel sistema, di servizio, di attesa, etc.)
  - Numeri medi (di utenti in coda, di chiamate presenti, di pacchetti persi o chiamate bloccate nell'unità di tempo, etc.)
- Di queste grandezze siamo in realtà interessati ad una più completa caratterizzazione statistica e quindi anche varianza o deviazione standard o anche distribuzione

## Come misurare le medie?

- Misurare grandezze medie richiede, come vedremo più avanti nel corso, molta attenzione
- Nel nostro simulatore abbiamo due possibilità:
  - raccogliere **tutti** i campioni delle grandezze cui siamo interessati, salvarli esternamente al simulatore (su file) e far svolgere l'analisi statistica delle nostre grandezze ad un programma esterno specializzato
  - effettuare all'interno del programma le adeguate operazioni matematiche per produrre in uscita le sole grandezze mediate
- Data la complessità e la durata delle nostre simulazioni, la soluzione tipicamente impiegata è la seconda

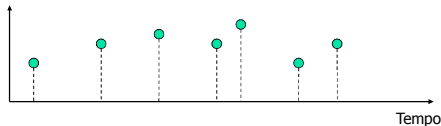
## Come misurare le medie?

- Le medie che vogliamo misurare sono di diverso tipo, a seconda delle grandezze coinvolte
  - medie puntuali
  - medie temporali
  - frequenze statistiche

## Medie puntuali

- Le usiamo per quelle grandezze di cui conosciamo i valori e il numero di occorrenze

$$\hat{x} = \frac{1}{N} \sum_{i=1}^N x_i$$



## Medie puntuali

- Nel codice
  - Accumuliamo i campioni  $x_i$   
`total += sample;`  
`nr_samples++;`
  - Al termine della simulazione calcoliamo  
`media = total/nr_samples;`
- In maniera analoga si può calcolare la varianza usando lo stimatore

$$\hat{\sigma}^2 = \frac{1}{N-1} \left( \sum_{i=1}^N x_i^2 - N \cdot \hat{x}^2 \right)$$

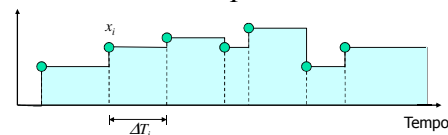
## Medie puntuali

- Esempi per un sistema a coda:
  - Ritardo medio di attesa
  - Numero di ritrasmissioni per pacchetto
  - Dimensione media del pacchetto

## Medie temporali

- Le usiamo per quelle grandezze che dipendono da funzioni del tempo
- Interessa il rapporto tra il valore che assumono e per quanto tempo tale valore viene mantenuto

$$\hat{\mu}_x = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T x(t) dt$$



## Medie temporali

- Nel codice
  - Accumuliamo le aree  $\Delta T_i \cdot x_i$   
`delta_time = current_time - last_time;`  
`total_area += sample*delta_time;`  
`sample = new_sample;`  
`last_time = current_time;`
  - Al termine della simulazione calcoliamo  
`media = total_area / final_time;`
- La valutazione è più onerosa perché i calcoli vanno fatti in coincidenza del campione successivo

## Medie temporali

- Esempi per un sistema a coda:
  - Numero medio di clienti nel sistema
  - Numero medio di server occupati

## Frequenze statistiche

- Le usiamo per valutare delle probabilità

$$\hat{p}_i = \frac{N_i}{N}$$

- Nel codice

- Contiamo i campioni favorevoli

```
if (condizione) { nr_good++; }  
nr_samples++;
```

- Al termine della simulazione calcoliamo

```
prob = nr_good / nr_samples;
```

## Frequenze statistiche

- Esempi per un sistema a coda:

- Probabilità di perdere un pacchetto
- Probabilità di consegnare un pacchetto in disordine (o in ritardo)
- Probabilità di trovare il servitore libero all'arrivo

## Caveat

- Le misure che effettuiamo in questo modo sono solo una stima, quasi certamente inaffidabile, delle grandezze che vogliamo misurare
- Non stiamo infatti tenendo in conto
  - l'effetto delle condizioni iniziali
  - l'effetto del transitorio
  - il fatto che quella che misuriamo è una singola realizzazione particolare del sistema simulato
- Durante il corso vedremo le procedure corrette per effettuare tali misure con un'adeguata garanzia della loro correttezza e precisione

## Inizializzazione

- Prima di iniziare la simulazione dobbiamo
  - inizializzare tutte le variabili necessarie per le misure
  - inizializzare le strutture dati necessarie alla simulazione
  - definire il valore di tutti i parametri caratteristici della simulazione, eventualmente tramite input dell'utente
  - mettere in moto la simulazione, schedando i primi eventi

## Input utente

- Per studiare il sistema vogliamo riprodurre il suo comportamento al variare di un certo numero di parametri
- Tenere questi parametri fissi all'interno del simulatore richiederebbe un diverso programma per ogni possibile loro combinazione
- È quindi normale prevedere l'inserimento di questi parametri
  - interattivamente
  - sulla riga di comando
  - tramite file di configurazione
  - con una combinazione dei metodi precedenti

## Terminazione

- Al termine dell'Event Loop, prima di uscire dal programma, è necessario comunicare all'utente i risultati della simulazione
- Se la simulazione è terminata regolarmente provvederemo a stampare (a schermo e/o su file) tutte le misure effettuate e ogni informazione che riteniamo opportuno fornire

## Simuliamo!

- Un'esecuzione del programma corrisponde una singola realizzazione del processo casuale che rappresenta il sistema simulato, ovvero un punto nello spazio dei possibili risultati
- Questo non ci basta: per caratterizzare e studiare il sistema vogliamo esplorare una parte significativa dello spazio dei risultati
- Pianifichiamo una **campagna di simulazione!**


## Simuliamo!

- L'esecuzione di una campagna di simulazione prevede di
  - differenziare i parametri di ingresso tra:
    - parametri fissati, della cui influenza non siamo interessati nella particolare campagna
    - parametri variabili, di cui vogliamo valutare l'effetto sulle prestazioni del sistema
  - identificare i parametri di uscita della cui variazione siamo interessati
  - eseguire simulazioni per ogni combinazione significativa dei parametri di ingresso variabili
  - aggregare e rappresentare le misure in uscita in maniera tale da rendere evidente le loro dipendenze dai parametri variabili d'ingresso

## Simuliamo!

- Parametri d'ingresso variabili
  - Per ognuno, occorre definire un intervallo di variazione ed il numero di valori in tale intervallo
  - Simulazioni per ogni combinazione di valori dei parametri
    - Per ogni combinazione, più simulazioni (con seme diverso e di lunghezza diversa) per ridurre l'effetto delle condizioni iniziali e delle sequenze pseudo-casuali
    - Complessità esponenziale con il numero di parametri: attenti a non esagerare!
- Rappresentazione dei risultati
  - Occorre scegliere la modalità migliore per evidenziare i nostri risultati -> famiglie di grafici parametrici

## Simuliamo!

- Rappresentazione dei risultati
    - Le misure ottenute dal nostro simulatore saranno quasi sempre usate per tracciare grafici parametrici in funzione dei parametri di ingresso
- 
- Scegliamo formati dei file di uscita che rendano la creazione di tali grafici semplice, almeno in relazione al programma che scegliamo di usare (gnuplot, Matlab, MS Excel, ...)