

Simulazione e Linguaggi di Programmazione

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.

Dal modello al programma

- La definizione e la specifica di un modello del nostro sistema da simulare non è sufficiente: occorre **eseguire** il modello
- Per modelli estremamente semplici tale operazione può essere eseguita manualmente, ma questo non è tipicamente il caso
- Occorre quindi convertire il modello in un programma da eseguire su un computer adeguato

Programmazione e simulazione

- Dopo la definizione di un modello del sistema da simulare, occorre convertire il modello in un programma da eseguire su un computer
- Diverse possibilità per passare dal modello al programma:
 - Linguaggi *special-purpose* per la simulazione
 - Linguaggi *general-purpose*
 - Ambienti integrati di sviluppo (eventualmente associati ad uno dei linguaggi precedenti)

Linguaggi *special-purpose*

- Si tratta di linguaggi di alto livello e fortemente orientati alla codifica di modelli di simulazione
- Offrono nativamente costrutti atti a rappresentare gli elementi propri del tipo di modellazione supportata:
 - Eventi discreti -> funzioni di creazione e scheduling degli eventi
 - Interazione processi -> funzioni di creazione e sincronizzazione dei processi

Linguaggi *special-purpose*

- Esempi di linguaggi dedicati alla simulazione:
 - SIMULA
 - GPSS/H
 - Arena/SIMAN
 - SIMSCRIPT II.5
 - ...
- Possono supportare un solo tipo di modellazione o entrambi

Linguaggi *special-purpose*

- Spesso si tratta di linguaggi di implementazione proprietaria
- Forniscono direttamente le funzioni necessarie per la generazione di variabili casuali e per la valutazione statistica dei risultati
- Possono generare codice intermedio in linguaggi *general-purpose* o interagire con funzioni scritte in tali linguaggi

Linguaggi *special-purpose*

- Caratteristiche
 - Nascondono al programmatore la complessità di gestione del sistema
 - Permettono di passare semplicemente dal modello al programma
 - Per i nostri scopi, si tratta di linguaggi estremamente specialistici

SIMULA: Esempio

```
Simulation Begin
Class FittingRoom; Begin
  Ref (Head) door;
  Boolean inUse;
  Procedure request; Begin
    IF inUse Then wait (door);
    inUse:= True;
  End;
  Procedure leave; Begin
    inUse:= False;
    Activate door.First;
  End;
  door:- New Head;
End;

Procedure report (message); Text message; Begin
  OutFix (Time, 2, 0); OutText (": " & message); OutImage;
End;
```

SIMULA: Esempio

```
Process Class Person (pname); Text pname; Begin
  While True Do Begin
    Hold (Normal (12, 4, u));
    report (pname & " is requesting the fitting room");
    fittingroom1.request;
    report (pname & " have entered the fitting room");
    Hold (Normal (3, 1, u));
    fittingroom1.leave;
    report (pname & " have left the fitting room");
  End;
End;

Integer u;
Ref (FittingRoom) fittingRoom1;

fittingRoom1:- New FittingRoom;
Activate New Person ("Sam");
Activate New Person ("Sally");
Activate New Person ("Andy");
Hold (100);

End;
```

SIMSCRIPT II.5: Esempio

```
Preamble
''
A simple telephone system model - CACI Products Company
files: TELPHN1.SRC

Normally mode is integer

Processes include
  GENERATOR

Every INCOMING.CALL has
  a CALL.ID

Define NUMBER.BUSY and
  LOST.CALLS
as integer variables
End ''Preamble
```

SIMSCRIPT II.5: Esempio

```
Main
  Activate a GENERATOR now
  Start simulation
  Print 1 line with LOST.CALLS thus
  15 phone calls were made and ** were lost due to
  busy lines
End 'Main

Process GENERATOR
  For I = 1 to 15 do
    Activate a INCOMING.CALL now
    Let CALL.ID(INCOMING.CALL) = I
    wait uniform.f (2.0, 6.0, 1) minutes
  Loop
End ''GENERATOR
```

SIMSCRIPT II.5: Esempio

```
Process INCOMING.CALL
  If NUMBER.BUSY < 2
    Add 1 to NUMBER.BUSY
    Wait uniform.f(6.0, 10.0, 2) minutes
    Subtract 1 from NUMBER.BUSY
  Else
    Add 1 to LOST.CALLS
  Endif
End ''INCOMING.CALL
```

Linguaggi *general-purpose*

- Dovendo il nostro modello essere codificato in un programma, qualsiasi linguaggio di programmazione può essere usato per la simulazione
- Non avremo tipicamente a disposizione nessun supporto nativo per rappresentare gli elementi del nostro modello
- Possiamo eventualmente ricorrere a librerie di funzioni *ad hoc*

Linguaggi *general-purpose*

- Dobbiamo essere in grado di realizzare tutte le funzioni di gestione necessarie al tipo di modellazione scelta
 - Eventi discreti → gestione e scheduling degli eventi
 - Interazione processi → creazione, sospensione e (ri)attivazione dei processi

Linguaggi *general-purpose*

- Se si immagina di mappare ogni processo su una funzione, le operazioni sospensione/riattivazione richiedono che il linguaggio supporti naturalmente il concetto di *coroutine* o *subroutine rientrante*, ovvero una funzione dotata di più punti di ingresso
- Questo non è vero per la maggior parte dei linguaggi di programmazione generale

Linguaggi *general-purpose*

- Abbiamo due possibilità per realizzare i processi interagenti del nostro modello:
 - creare o usare un'infrastruttura di programmazione che permetta di emulare le funzioni rientranti, in pratica tramite la realizzazione in piccolo di un linguaggio *special-purpose*
 - utilizzare le funzionalità di multi-processing/multi-threading del sistema operativo ospite

Linguaggi *general-purpose*

- La codifica di modelli basati sullo scheduling di eventi discreti non rivela requisiti particolari: sono tipicamente sufficienti le funzionalità e le strutture dati native del linguaggio
- Occorre realizzare le funzioni di gestione degli eventi e di controllo della simulazione, ma si tratta di codice facilmente riusabile

Linguaggi general-purpose

- **Caratteristiche**
 - Mostrano al programmatore la complessità di gestione del sistema
 - Compilatori più facilmente disponibili
 - Controllo puntuale su ogni elemento della simulazione
 - Non adeguati per simulare processi interagenti
 - Richiedono la codifica di ogni elemento della simulazione, moltiplicando le possibilità di errore

Ambienti integrati di sviluppo

- Si tratta di tool, spesso grafici, che permettono di simulare sistemi molto complessi
- L'utente ha a disposizione ampie librerie di blocchi elementari da comporre fino a costruire il modello del proprio sistema
- Il sistema si preoccupa di gestire l'interazione tra i blocchi e l'intera simulazione

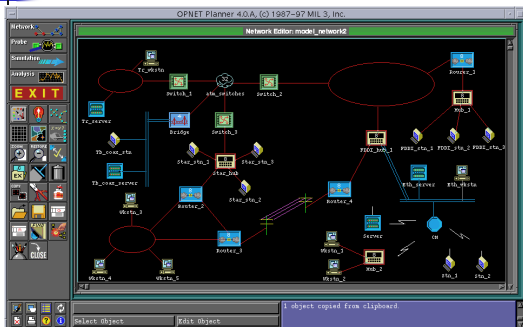
Ambienti integrati di sviluppo

- Ogni blocco può essere pre-programmato oppure costituire un contenitore per codice scritto dall'utente
- Ogni tool è di solito dedicato ad un settore specifico: eventuali librerie arricchiscono le funzionalità ma non lo spettro di applicabilità (almeno non a settori completamente diversi)

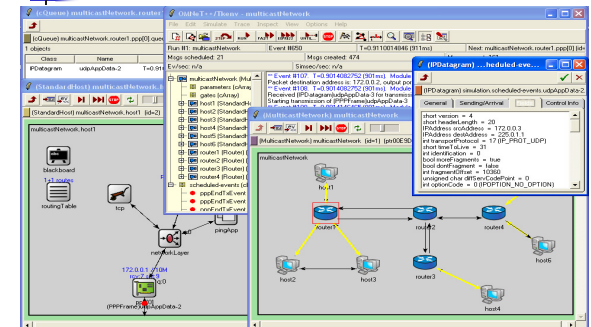
Ambienti integrati di sviluppo

- **Caratteristiche**
 - Creazione rapida del programma di simulazione
 - Possibile simulare senza saper programmare
 - Librerie di modelli molto ricche
 - La complessità della simulazione è nascosta
 - Occorre fidarsi della qualità dei modelli di libreria
 - Non si ha il controllo su numerosi aspetti della simulazione: si rischia di trascurare informazioni chiave, quali la precisione dei risultati ottenuti
 - La programmazione di blocchi personalizzati può essere complessa
 - Il programma ottenuto può non essere efficiente

OpNet



OMNeT++





- Popolare ambiente integrato (non grafico)
- Orientato alla simulazione di Internet
 - Ricca libreria di blocchi dedicati alla simulazione di TCP/IP, routing e protocolli multicast, in reti cablate e wireless (LAN e satellitari)
- Programmato in linguaggio C++ e OTcl