Brevi note sul progettino di "Advanced programming in C++"

Il comando per fare partire la simulazione dall'istante k all'istante h ha la sintassi:

```
start   [ h ]   k   [ -d | -D ]
```

dove h e k sono due numeri interi positivi o nulli (il default per h è 0).
Se si specifica l'opzione –d, per ogni istante viene chiamata la funzione di stampa a video per ogni blocco.

Esiste una libreria di blocchi standard (ossia immediatamente utilizzabili dall'utente) già popolata, il cui contenuto può essere visionato direttamente guardando la definizione delle classi nel modulo "block", oppure in maniera più veloce ed intuitiva visionando il file di configurazione "standard_blocks.txt". Esso contiene:

- nella prima colonna di tale file ci sono i nomi dei blocchi standard,;
- nella seconda si specifica se il blocco è con o senza stato (se ciò è implicito va messo "-");
- nella terza colonna un stringa di quattro lettere indicante il tipo di dimensionalità dell'ingresso e dell'uscita del blocco; la sintassi è
  { NI | SI | MI } { NO | SO | MO }
  dove NI = No-Input, SI = Single-Input, MI = Multi-Input, e analogamente per gl output;
- nella quarta colonna il numero di parametri che il costruttore del blocco (nel file di ingresso) deve accettare;

L'applicazione è strutturata in un certo numero di moduli che interagiscono tra loro mediante le rispettive interfacce.
La classe principale di ogni modulo, essendo presente in un'unica istanza, è assimilabile ad un *agente*. Questa è la filosofia di progetto: agenti che si passano i dati l'un l'altro fin quando non sono stati eseguiti tutti compiti. Per quanto è stato possibile, si è cercato di mantenere alta l'indipendenza tra i moduli.

Il progettino è stato inoltre pensato in modo da rendere il più semplice possibile l'aggiunta di nuovi blocchi standard. I passi da seguire sono:

1) aggiungere una riga al file "standard_blocks.txt"
2) coerentemente con il punto 1, definire la relativa classe nel modulo "block" specificando il suo comportamento con la derivazione dalle classi del secondo livello della gerarchia (vedere le classi già esistenti per esemplificazione); alcune funzioni funzioni membro si possono e si devono semplicemente copiare dalle altre classi già fatte; le uniche funzioni che bisogna progettare (le cui firme sono tuttavia predefinite) sono: a) il costruttore del blocco (che in realtà si scrive sempre nello stesso modo), b) la funzione "refresh()" che effettua il passo di aggiornamento del blocco, c) le funzioni

"initOperation()" e "endOperation()" che eventualmente specificano le operazioni di inizializzazione e chiusura della memoria privata della classe, le quali vengono richiamate rispettivamente all'inizio ed alla fine di ogni simulazione, d) una funzione di stampa per il blocco;

3) aggiungere una riga alla funzione "BlockBuilder::build" nel modulo "block" che effettua il demultiplexing dei costruttori.

La gerarchia esistente fa sì che la classe così definita abbia già incapsulati dentro di sé i comportamenti che le si vogliono dare.

E' possibile definire, in file separati, dei tipi superblocco (non parametrici), ossia degli agglomerati di blocchi standard, connessi tra loro Il nome di tali file è lo stesso del nome del superblocco con estensione ".diss".
Da un punto di vista pratico i tipi superblocco si utilizzano esattamente come i blocchi standard.
Un superblocco può contenere al suo interno altri superblocchi, e questi ultimi possono contenerne ancora altri, e così via.

I file di definizione di superblocco vengono analizzati una sola volta, ma i superblocchi possono essere utilizzati un numero qualsiasi di volte, perché si salva in memoria una loro rappresentazione, in una opportuna tabella.

Dopo la fase di analisi sintattica dell'input, i vari modelli che costituiscono la rappresentazione di un superblocco, e i superblocchi stessi (cioè le istanze di classe SuperBlock che vengono via via generate) vengono buttati via perché inutili per la simulazione. Tuttavia essi sono assolutamente necessari per costruire incrementalmente i vari modelli, e per effettuare le varie clonazioni.
In ogni caso si potrebbe espandere il programma aggiungendo l'interfaccia grafica, ed in tal caso si potrebbe fare un uso intensivo delle varie istanze di classe SuperBlock (la classe non è stata neanche completamente implementata, è stato implementato solamente lo stretto necessario).

Dunque prima dell'ordinamento si ha una lista di soli blocchi standard connessi fra loro (niente superblocchi).
L'algoritmo di ordinamento dei blocchi (per trovare l'ordine di aggiornamento degli stessi) è una sorta di ordinamento topologico del grafo finale.

Dopo l'ordinamento viene lanciata la console.

Ovviamente queste note sono troppo brevi per spiegare tutte le scelte di progetto, con relativi vantaggi e svantaggi, tuttavia speriamo che sia sufficientemente chiaro e che dia un'idea generale.

Per questioni di tempo, purtroppo, non ci è stato possibile fare un testing esaustivo, tuttavia tutto quello che viene qui specificato (dovrebbe) funziona(re) perfettamente.

Restiamo a disposizione per qualunque chiarimento/segnalazione, e naturalmente per il continuo dell'esame.

Andrea Bravi
Vincenzo Maffione