ASUPRA IMPLENTĂRII UNUI SIMULATOR GENERIC

GABRIEL DIMITRIU
Axway Romania

Rezumat: Autorul propune o nouă abordare a simulatorarelor generice pe baza formalismului DEVS modificat ca o continuare la implementarea începută teoretic în lucrarea FLOREA I.L. *Upon Multiservers Preemptive Queueing System: A Formal and Object Oriented Approach*. Această abordare a fost implementată sub forma unei biblioteci de simulare orientată pe obiecte care să permită utilizatorilor derivarea şi implementarea unor noi modele pentru sisteme care nu au funcțional nimic în comun cu excepția interacțiunilor între ele, fără a modifica o implementare existentă. În prezent a fost folosită în special pentru simularea sistemelor de aşteptare.

Cuvinte cheie: DEVS, general purpose simulator,GPSS,SIMUB

1. INTRODUCERE

Simularea unitară a unor sisteme foarte complexe şi variate compuse din interacțiuni, legi fizice cu ajutorul unui singure biblioteci de funcții (pachet de clase) reprezintă o mare provocare. Această bibliotecă (Application Programming Interface – API) trebuie sa aibă capacitatea de a evolua în timp fară a schimba structura ei. În acelaşi timp trebuie să dispună de suficientă mobilitate pentru a încorpora în ea într-un viitor apropiat inteligență artificială şi să permită interconectarea cu alte programe de simulare.

Programul care realizează modelarea şi simularea unui sistem se compune din trei obiecte de bază: *sistemul real*, care este văzut ca o sursă de date; *modelul*, care este o colecție de instrucțiuni de generare a datelor comparabile cu sistemul real; *simulatorul*, care este o colecție de instrucțiuni realizate într-un limbaj care execută instrucțiunile din model.

Pentru modelarea elementelor sistemului a fost ales formalismul DEVS (Discrete Event System Specification) [1, 2, 3] deoarece acest formalism permite cea mai mare flexibilitate în modelarea sistemelor. Acest formalism, prezentat în lucrarea [4], a fost modificat pentru a putea modela cît mai multe tipuri de probleme.

1.1. DEFINIREA FORMALĂ A MODELULUI DE BAZĂ DEVS

Orice sistem real poate fi considerat ca un sistem dinamic de timp real, componentele sale fiind descrise de trei tipuri de variabile: *variabile de intrare*, *variabile de stare* şi *variabile de ieşire*. În general acest model poate fi generalizat oricărui sistem real care are o interacțiune cu alt sistem, caz în care se spune ca are o tranziție de tip extern şi are o prelucrare internă în momentul în care nu are

interacțiuni cu alte sisteme, numită şi tranziție internă. Aceste două interacțiuni vor fi definite de către o funcție de traziție externă, respectiv internă. Asupra tranzițiilor nu sunt restricții.

Descrierea dinamică a modelui DEVS [1, 2, 3, 10, 11] este următoarea:

$$DEVS = (X, S, Y, ta, \delta_{\text{int}}, \delta_{ext}, \lambda) \quad (1)$$

unde:

- X este mulțimea de "intrare", este produsul cartezian al variabilelor de intrare ;
- Y este mulțimea de "ieşire", este produsul cartezian al variabilelor de ieşire ;
- S este mulțimea stărilor, este produsul cartezian al variabilelor de stare ;
- $ta : S \to R_+ \cup (0, \infty)$ este funcția de avans a timpului, valoarea sa *ta(s), s∈S* reprezintă intervalul de timp după care sistemul face o tranziție între starea *s* şi starea $\delta_{int}(s)$;
- $\delta_{\text{int}}(s) : S \to S$ este funcția de tranziție internă a sistemului; această funcție reprezintă tranziția între două stari fară influență externă ;
- $\delta_{ext}(s) : Q \times I \to Q$ unde $Q = \{(s, e) \mid s \in S, e \in R, 0 \le e \le ta\}$ este funcția de tranziție externă, această funcție reprezintă tranziția între două stări în cazul în care la momentul de timp *e* a venit un eveniment extern ;
- $\lambda : S \to Y$ este funcția de ieşire, dacă sistemul este în starea *s* atunci λ(*s*) va fi informația care se va transmite la ieşire înainte ca sistemul să facă tranziția sa (internă sau externă).

Din această caracterizare a modelului DEVS avem următoarea funcționare a modelului de bază :

$$\exists e \in [T, T + ta(s)] \Rightarrow \exists x \in X \quad \text{se execută } \delta_{ext}(s, e, x) \quad (3)$$

$$\forall e \notin [T, T + ta(s)] \Rightarrow \text{se execută } \delta_{\text{int}\,t}(s) \quad (4)$$

1.2. DEFINIREA FORMALĂ A MODELULUI DE REȚEA CU STRUCTURĂ STATICĂ

Cuplarea mai multor modele se face utilizând un nou model, numit *model de rețea* care poate fi cu structură statică sau variabilă. În cazul implementării mele este un model cu structură static-variabilă, deoarece are capabilitatea de modificare dinamică a structurii sale interne. Acest model este compus din mulțimea modelelor de bază sau de rețea care formează elementele interne ale modelului rețea şi din mulțimea legăturilor şi a porturilor proprii rețelei, precum şi structurile necesare simulării.

Această modelare permite organizarea rețelei în timp real în funcție de necesități, deorece nu se pierd informații în cazul reorganizării, ci se vor adăuga noi intrări în listele dinamice care memorează legăturile dintre elementele constituente ale rețelei. Elementele unei rețele pot fi pasive, caz în care nu au tranziție internă, şi active, caz în care au tranziție internă. Aceste elemente sunt ținute în două cozi de

procesoare active, respectiv pasive. Când este primit un mesaj la unul din porturile rețelei, acest mesaj este rutat mai departe către elementele care interacționează cu acest port. Această rutare este efectuată de către submodelul de interconectare.

Submodelul de interconectare este un model care nu face altceva decât reține conexiunile între elementele ce constituie rețeaua şi efectuează rutarea mesajelor.

Rutarea mesajelor poate fi efectuată în trei moduri: primul mod de funcționare este modul "câştigatorul ia totul", în care un mesaj poate fi rutat către un singur element intern care este liber; al doilea mod de funcționare este modul de distribuție a resurselor, în care mesajul este furnizat cîstigătorului, dar restul de mesaj care a mai rămas este dat celui de-a doilea element; al treilea mod de funcționare este modul duplicat, în care toate elementele care sunt legate la acel port primesc o copie a mesajului respectiv.

Al doilea mod de funcționare este un mod de particular în care se cunosc necesitățile fiecărui element. Dacă elementul care are necesitatea egală cu disponibilul, are prioritatea necesară şi este liber el va primi tot mesajul. Dacă nu există nici un element care sa aibă necesitatea egală cu disponibilul sau acel element nu este liber, elementul declarat câştigător ia din disponibil numai cât are nevoie iar restul este împărțit din nou către elementele rămase.

2. DESCRIEREA IERARHIEI CLASELOR

Pentru a crea un simulator generic de genul Simulink vom avea nevoie de un model generic care poate fi îmbunătățit în timp, fără a aduce modificări bibliotecii de simulare. Aceasta a fost realizată utilizînd un limbaj orientat pe obiecte, foarte flexibil şi foarte rapid. Toate aceste caracteristici recomandă limbajul C++, iar pentru facilitatea de *serializare* a fost ales mediul de dezvoltare Visual C++ de la Microsoft, bazat în această prima versiune a simulatorului pe tehnologia MFC.

Arhitectura sistemului şi implementarea este complet diferită de cea prezentată în [4] deoarece arhitectura sistemului a fost gîndită pentru maximum de flexibilitate.

Simulatorul Generic a fost implementat până în acest moment în patru mari module, având o mare disponibilitate de extindere: o librărie (modul) pentru generarea repartițiilor [7], o librărie (modul) pentru parsarea funcțiilor matematice [8], o librărie (modul) central de simulare şi un modul de validare, testare şi în acelaşi timp funcționând ca interfață grafică de simulare.

2.1. CLASE DE INTRARE-IEŞIRE

Aceste clase sunt *CGInput*, *CGOutput* respectiv *CInput*, *Coutput* şi reprezintă porturile de intrare/ieşire pentru toate sistemele: procesoare şi rețele. *CInput* ține un pointer către clasa generator şi de asemenea un fişier ataşat acestui port pentru log (jurnalizare). Dimensiunea portului este arbitrară. Portul de intrare poate să conțină câte un generator pentru fiecare element, generator de zgomot care se suprapune peste semnalul original.

2.2. CLASA DE TRANZIȚIE

Aceasta clasă este *CGTransition* şi modelează funcționarea reală a unui model de bază (procesor) şi trebuie să fie derivată de fiecare utilizator pentru a modela o aplicație specifică. Această clasă are un generator intern de numere aleatoare şi un parser de funcții matematice. Dacă modelul este suficient de simplu pentru a fi modelat de o funcție matematică de intrare-ieşire atunci această clasă nu trebuie derivată, deoarece parserul este definit pe produsul cartezian format din mulțimea intrărilor cu mulțimea variabilelor intermediare cu valori în produsul cartezian format din mulțimea ieşirilor cu mulțimea valorilor intermediare.

Există însă şi unele procesoare care nu apelează această clasă în special procesoarele cu destinații speciale, de exemplu cele de distribuție de date, cele care fac calcule sau multiplexoarele şi demultiplexoarele.

2.3. CLASA DE PROCESARE

Această clasă este *CGProcessor* cu derivatele sale: CGMesure pentru realizarea de măsurători, CGBus pentru transmiterea pe bus de mesaje, CGDistributionQ, CGDistributionS şi CGDistributionBus care sunt clase de distribuție a datelor de intrarea S care este secvențială, Q este pe baza unei cozi interne (utilizarea acestora va fi prezentată în secțiunea (3.1)) iar Bus este de fapt un multiplexor/demultiplexor. Această clasă conține modelul pentru procesorul activ sau pasiv. Ea implementează coada de mesaje şi comportamentul procesorului, exceptând tranzițiile pe care le apelează din clasele derivate din *CGTransition*. Această clasă va fi derivată numai dacă se schimbă comportamentul procesorului, are altă politică de întreruperi şi de cozi față de procesorul generic şi față de cele 4 variante ale lui. Procesorul generic are implementată politica de cozi cu sau fară prioritate descentralizată.

2.4. CLASA REȚEA

Această clasă este *CGNetworkS*, fiind clasa de bază care conține toate informațiile asupra sistemului de simulat. Ea conține conexiunile interne între procesoare şi rețele, tot aici sunt informațiile despre procesoare: care dintre ele sunt pasive şi care sunt active.

Tot în această clasă se află implementat şi coordonatorul dinamic al rețelei care este în lucru în momentul scrierii acestui articol. Acest coordonator este specific fiecărei rețele şi este bazat pe un server de evenimente. Acest coordonator este baza pentru următoarea fază a dezvoltării acestui simulator, şi anume distribuirea lui pe diferite sisteme, deoarece în acel moment fiecare rețea poate fi asignată spre prelucrare unei maşini diferite, legăturile dintre ele realizându-se prin socket-uri, socket-uri care vor înlocui funcția de copiere din clasele de intrare/ieşire.

3. DESCRIEREA PROBLEMELOR IMPLEMENTATE

Până în prezent au fost implementate probleme de aşteptare şi câteva probleme de test cu contoare de evenimente. Toate cozile care sunt implementate sunt de tipul dinamic cu capacitate infinită. Pentru mai multe versiuni de test se poate vedea [6] care este manualul de validare a simulatorului generic. Toate acele versiuni de test sunt implementate în librăria de verificare/validare a simulatorului pentru a putea fi rulate în aceleaşi condiţii de fiecare dată când sunt efectuate modicări asupra bazei simulatorului.

3.1. SISTEM DE AŞTEPTARE CU M STAŢII IN PARALEL

Să considerăm un sistem care are în compunere **m** staţii de servire în paralel este bine cunoscut în lumea reală ca sistem de deservire de la casă sau de la un service auto. Dacă există o staţie liberă atunci clientul este deservit de acea staţie şi timpul de deservire este o selecţie a unei variabile aleatoare de o distribuţie cunoscută. Dacă toate staţiile sunt ocupate atunci clientul este pus într-o coadă de aşteptare de tip FIFO (First In First Out) şi un timp de aşteptare este contorizat pentru acest client. Timpul de sosire între doi clienţi este o variabilă aleatoare de distribuţie cunoscută. Dacă o staţie devine liberă dar nu există nici un client în sistem pentru a fi deservit de către această staţie atunci staţia devine *idle* şi un timp *idle* este contorizat.

Se vor face următorele notaţii: **nr_clients** numărul de clienţi care au fost procesaţi, **waiting_clients** sunt numărul clienţilor în aşteptare, **tid** timpul de *idle* al maşinilor, iar **tst** este timpul de service al maşinilor.

Sunt posibile două implementări în funcţie de locaţia unde este implementată coada mesajelor în sistem: coada este implementată în procesoare (staţii) şi coada este implementată în sistem.

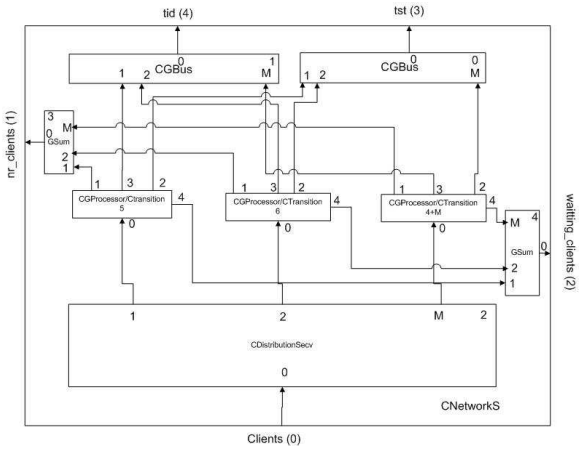

Figure 1 : Sistem de aşteptare cu m staţii

3.1.1 IMPLEMENTAREA COZII ÎN STAȚII

În prima versiune fiecare stație stă într-o coadă circulară, aceasta este introdusă în coadă la final în momentul în care stația primește un client pentru prelucrare.

În figura 1 este prezentată această implementare pentru m = 3 si următoarele setări pentru stații: stația 0 are 2.0 unități *delay* (întârziere), stația 1 are 2.5 unități *delay* și stația 3 are 3.0 unități *delay*. Pentru simulare s-a folosit precizia de ceas de 0.5 unități.

T	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Clients	0	1	1	1	-	1	1	1	-	-	-
nr_clients	0	0	0	0	0	1	1	2	2	4	4
waiting_cl	0	0	0	0	0	0	0	0.5	0.5	1.5	1.5
Tid	0 0 0	0.5 0.5 0.5	0.5 1 1	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	1 1 1.5
Tst	0 0 0	0 0 0	0.5 0 0	1 0.5 0	1.5 1 0.5	2 1.5 1	2.5 2 1.5	3 2.5 2	3.5 3 2.5	4 3.5 3	4 3.5 3

T	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10
Clients	1	1	1	-	-	-	-	-	-	-
nr_clients	4	5	5	5	7	7	8	8	8	8
waiting_clients	1.5	1.5	1.5	1.5	2.5	2.5	2.5	2.5	2.5	2.5
Tid	1.5 1 1.5	1.5 1 1.5	1.5 1 1.5	1.5 1 1.5	1.5 1 1.5	2 1 1.5	2.5 1 1.5	3 1.5 1.5	3.5 2 1.5	4 2.5 1.5
Tst	4 4.5 4	4.5 5 4.5	5 5.5 5	5.5 6 5.5	6 6.5 6	6 7 6.5	6 7.5 7	6 7.5 7.5	6 7.5 8	6 7.5 8.5

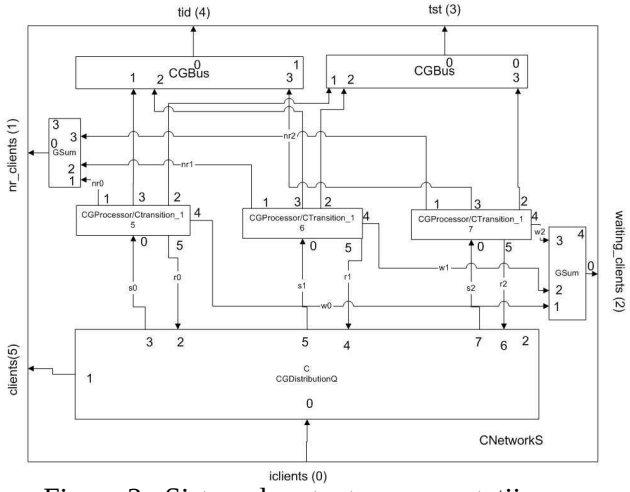

Figure 2 : Sistem de aşteptare cu m stații

3.1.2 IMPLEMENTAREA COZII ÎN SISTEM

În a doua versiune fiecare stație stă într-o coadă circulară, aceasta este introdusă în coadă la final, în momentul în care stația a terminat de prelucrat un client.

În figura 2 este prezentată această implementare pentru m = 3 şi următoarele setări pentru stații: stația 0 are 2.0 unități *delay*, stația 1 are 2.5 unități *delay* şi stația 3 are 3.0 unități *delay*. Pentru simulare s-a folosit precizia de ceas de 0.5 unități.

3.2. SISTEM DE AŞTEPTARE CU COADĂ DE PRIORITĂȚI

Clienții care intră în sistem sunt sortați după prioritate în **n** clase numerotate 1,…,n. Clienții care au prioritatea **i** au prioritate față de clienții care au prioritatea j>i. Clienții care au aceaşi prioritate sunt serviți în ordinea sosirii.

Dacă un client cu prioritate mică soseşte şi toate stațiile sunt ocupate cu deservirea clienților cu prioritate mai mare şi cozile de intrare cu prioritate mai mică sau egală cu a clientului sosit sunt goale el va aştepta pâna când o stație devine liberă.

T	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
Clients	0	1	2	3	-	4	5	6	-	-	-
nr_clients	0	0	0	0	0	1	1	2	2	4	4
cl_waiting	0	0	0	0	0	0	0	0.5	0.5	1.5	1.5
Tid	0 0 0	0.5 0.5 0.5	0.5 1 1	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 1.5	0.5 1 2
Tst	0 0 0	0 0 0	0.5 0 0	1 0.5 0	1.5 1 0.5	2 1.5 1	2.5 2 1.5	3 2.5 2	3.5 3 2.5	4 3.5 3	4.5 4 3

T	5.5	6.0	6.5	7.0	7.5	8.0	8.5	9.0	9.5	10
Clients	7	8	9	-	-	-	-	-	-	-
nr_clients	4	5	6	6	6	6	9	9	9	9
cl_waiting	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
Tid	0.5 1 2.5	0.5 1 2.5	0.5 1 2.5	0.5 1 2.5	0.5 1 2.5	0.5 1 2.5	0.5 1 2.5	1 1.5 3	1.5 2 3.5	2 2.5 4
Tst	5 4.5 3	5.5 5 3.5	6 5.5 4	6.5 6 4.5	7 6.5 5	7.5 7 5.5	8 7.5 6	8 7.5 6	8 7.5 6	8 7.5 6

Dacă cozile cu prioritate mai mică sau egală cu prioritatea clientului sosit nu sunt goale, clientul intră în coadă pe ultima poziție cu prioritatea sa şi un timp de aşteptare este contorizat pentru aceast client.

Dacă o stație devine liberă şi coada nu este goală ea va deservi primul client cu cea mai mică prioritate din coadă.

Dacă în schimb stația devine liberă şi coada este goală, această stație devine *idle* şi un timp de *idle* este contorizat pentru acea stație şi stația intră pe ultima poziție în coada de *idle*.

În schema bloc de mai jos avem următoarele ieşiri ale sistemului: **tst** timpul de service, **nc** este numărul de clienți intrați în sistem, **evs** este numărul de evenimente simulat la un moment dat, **twt** este timpul de aşteptare al clienților, **tst** este timpul de service al clienților, **ns** este numărul de clienți deseviți iar **news** este numărul total de evenimente simulate.

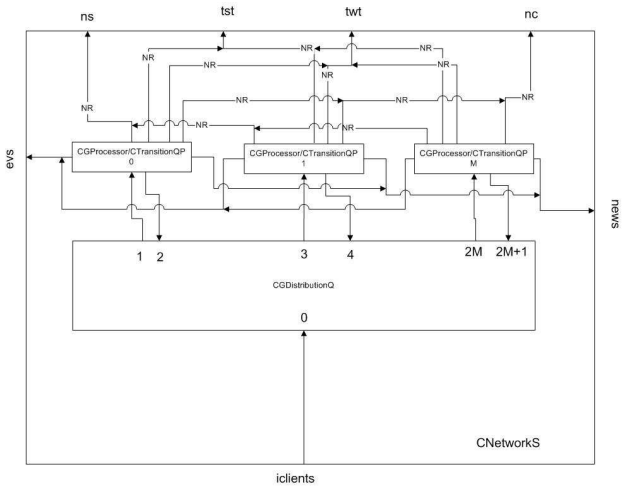

Figure 3: Sistem cu m staţii cu priorităţi

Următorul tabel prezintă rezultatele simulării unui sistem cu trei maşini cu următorii timpi de prelucrare: 2, 2.5 respectiv 3 unităţi. Simularea a fost efectuată cu precizia de ceas de 0.5 unităţi.

T	0.5	1.0	1.5	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7
iclients	1	2	1	1	2	1	-	-	-	-	-	-	-
Evs	0	0	0	1	1	2	2	4	4	4	5	6	6
Nc	1 0	1 1	2 1	3 1	3 2	4 2	4 2	4 2	4 2	4 2	4 2	4 2	4 2
news	0	0	0	1	1	2	2	4	4	4	5	6	6
ns	0 0	0 0	0 0	1 0	1 1	1 1	1 1	3 1	3 1	3 1	4 1	4 2	4 2
twt	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 1.5
tst	0 0	0 0	0 0	2 0	2 0	2 2.5	2 2.5	7 2.5	7 2.5	7 2.5	9.5 2.5	9.5 4.5	9.5 4.5

4. CONCLUZII

În acest articol au fost prezentate numai o parte din modalitățile de lucru ale noului nucleu de simulare generică. În acest moment sunt în lucru noi metode de a implementa coordonatorul distribuit şi de a compartimenta sistemul de simulare pentru a putea simula cât mai multe tipuri de probleme precum şi migrarea de la Visual C++ 6.0 la Dot Net.

BIBLIOGRAFIE

1. Chisman, J., *Introduction to Simulation and Modeling using GPSS/PC*, Minuteman Software, (1989).
2. Ion Văduva, *Modele de Simulare*, Editura Universității din Bucureşti, Bucureşti, ISBN 973-575-862-8 (2004).
3. Fishman,G.S., *Principles of Discrete Event Simulation*, Wiley, New York, (1978)
4. Florea I.L., *Upon Multiservers Preemptive Queueing System: A Formal and Object Oriented Approach*.
5. G. Dimitriu, *Implementation Manual for Generic Simulator*, 2006
6. G. Dimitriu, *Testing Manual for Generic Simulator*, 2006
7. https://sourceforge.net/projects/librepartitions
8. https://sourceforge.net/projects/parserfunction
9. Zeigler B.P., *Theory of Modelling and simulation*, John Wiley,1976.
10. Zeigler B.P., *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, 1984
11. Zeigler B.P., Praehofer H, Kim T.G., *Theory of Modelling and Simulation Interpreting Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000

Abstract: The author proposes a new implementation of general purpose simulators based on a modified DEVS formalism. This simulator is implemented as an object oriented library. This implementation offers to the users a method to implement new models with functionalities that don't have anything in common, except the interactions, without modify the library, or the implementation of an existed model. Currently, it was used to implement waiting systems and some event counter for testing purpose.