Chapitre 7
Les Outils de la Simulation
(Logiciels et langages de simulation)

7.1.Introduction.

Les modèles de simulation peuvent être programmés dans une variété de langages :

A. D'abord, on peut utiliser les langages de universels (BASIC, FORTRAN, PASCAL, C, Java et leurs variantes…). Ils sont dotés de générateurs de nombres au hasard. Il faut cependant programmer soi-même la plupart des procédures de simulation (gestion de l'échéancier, mise en file, tirage de variables et processus aléatoires, mesures et statistiques, édition). Certains de ces langages de programmation, du moins dans leur variantes actuelles comportent des bibliothèques spécialisées : Exemples bibliothèques C/C++ (CNCL, C++SIM).

B. Il existe des langages dédiés simulation orientée qui peuvent être généralistes ou spécialisés vers la résolution de tel ou tel problème: QNAP, QNA, GPSS (General Purpose Digital Simulator), SIMSCRIPT, SIMULA, GASP, SIML, SOL, CSL, OSSL....1001, LISP, MOSIM, SLAM, SIMAN,… Ils fournissent une aide pour la spécification des problèmes et l'écriture des programmes. Le développement du modèle de simulation est réalisé par un programme écrit par l'utilisateur à partir de primitives de modélisation offerte par le langage. Ils peuvent comporter par exemple des fonctions de type « mise «en file d'attente », « gestion de priorités »…..Ils offrent la possibilité d'intégrer des procédures écrites dans des langages évolués FORTRAN, PASCAL, C et leurs variantes…). La programmation dans ces langages concerne les aspects suivants :

(i) **le modèle ou logique du système** : imitation de la dynamique du système (son comportement) compte tenu de l'environnement. (voir exemple du modèle de file d'attente traité en TD. Dans le modèle traité ici, la seule différence est que la loi de service est normale et qu'on s'intéresse à d'autres mesures de performance : temps de réponse).

(ii) **Données** : Représentent les paramètres du système à simuler (attributs…).

(iii) **Animation** : Représentation graphique du système. Elle permet surtout la présentation du modèle à l'utilisateur en lui épargnant les détails ennuyant de la modélisation.

(iv) **Statistiques** : Les résultats sont rapportés en un ensemble de statistiques sur la base desquelles on peut comparer les différents scénarios (comparaisons d'une nouvelle conception par rapport à l'ancienne) et permettent ainsi une aide à la décision.

C. Outils intégrant un éditeur graphique (exemple : Comnet, Opnet, Modline, Ses Workbench , Arena, Automod).

Il faut noter que les langages spécialisés et environnements fournissent des outils de haut niveau, mais au détriment de la flexibilité des langages universels. Dans les langages de simulation usuels, on utilise souvent les langages universels pour programmer les aspects complexes du modèle et les opérations non prévues dans le langage spécialisé. Un autre inconvénient des langages spécialisés : il faut l'apprendre. C'est un investissement en temps non négligeable, particulièrement pour des usages occasionnels à cause du fait que ces langages ont leur propre syntaxe et sémantique (souvent excentrique).

D. Certains environnements de simulation de haut niveau proposent une approche de « non-programmation » où les modèles sont spécifiés par de simples clics en utilisant des boîtes à outils ou en en manipulant des objets graphiques sur l'écran de l'ordinateur comme dans les jeux. Cette approche est très commode pour construire des modèles par un utilisateur non initié avec des cadres préprogrammés (**exemple** : les simulateurs de vols, de conduite, etc…). Cependant, l'inconvénient majeur est que l'utilisateur a très peu de possibilités d'agir sur le simulateur. Les erreurs éventuelles (par exemple dans les résultats numériques d'évaluation de performances et autres sont difficiles à détecter, à moins de connaître avec précision le langage source.

Le but de ce paragraphe n'est pas de faire la promotion de tel ou tel langage, ni de faire une étude comparative, ce qui serait très difficile, voire impossible. Un tableau comparatif des langages et logiciels de simulation à évènements discrets pourra être trouvé par exemple sur le site http://www.tbm.tudelft.nl/webstaf/edwinv/SimulationSoftware/index.htm. L'objectif de ce paragraphe est de donner un aperçu sur ces différents langages. Certains sont librement téléchargeables sur le Web, d'autres commercialisés. Les sites indiqués ici donnent des indications sur les expériences d'utilisation lorsqu'elles existent et fournissent souvent des démonstrations avec des versions pédagogiques. Il faut noter que les expériences de certains langages se limitent à des recherches universitaires, alors que d'autre ont des expériences plus vastes dans les domaines de l'économie ou de l'industrie. Le second objectif de ce paragraphe est surtout de montrer comment les différentes notions théoriques des chapitres précédents sont utilisées dans ces langages. Notons que ce document contient une description des langages usuels qui semblent être le plus enseignés dans certains cours de simulation, parfois même utilisés dans certains secteurs de l'économie et de l'industrie. Ils reprennent tous le même exemple didactique : la file d'attente à un serveur (M/M/1 ou parfois M/G/1).

7.2. GPSS (General Purpose System Simulation). Langage orienté vers les systèmes de files d'attente tels que les modèles d'ateliers ou jobshops), réalisé initialement par IBM en 1961. Depuis, il existe plusieurs nouvelles versions implémentées et améliorées (GPSS/H, GPSS/World). GPSS/H comporte un des fonctions mathématiques, des générateurs de nombres et variables aléatoires, un animateur graphique(Proof Animation ^{TM}). Le diagramme des modules est un moyen commode de décrire le système à simuler (environ 40 blocs dans GPSS). Les entités appelées transactions (objets, clients, unités de trafic, machines) sont caractérisées par des attributs (lois d'arrivées, de service, de panne, et/ou leurs moyennes) et reliées entre elles peuvent être vues comme un flux à travers le diagramme. (**http://cours.polymtl.ca/ifsos/GPSS/GPSS/).**

Modules/Attributs	Description du module ou de l'attribut
GENERATE	Génère l'arrivée d'un événement (entité) spécifié par

	RVEXPO(1,&IAT)
RVEXPO(1,&IAT)	Variable exponentiellement distribuée de moyenne &IAT
QUEUE*	Début de collecte des données du temps de réponse spécifié par SYSTIME (entrée d'une transaction dans le système)
SYSTIME	Attribut de QUEUE
QUEUE	Début de la collecte des données de la file
DEPART	Fin de collecte des données sur le temps de réponse.
SEIZE	caractérise la ressource pour m=1 (STORE si m¥1) serveurs
CHECKOUT	Attribut spécifiant la possession de la ressource
ADVANCE	Module spécifiant le délai de transaction auprès de la ressource
RVNORM(1&MEAN,&STDV)	Variable normale de moyenne &MEAN et d'écart-type &STDV
RELEASE	Fin d'utilisation de la ressource CHECKOUT (RETURN pour m¥1)
DEPART	Fin de collecte des données du temps de réponse SYSTIME
TEST	Teste si le temps de séjour M1¥4 minutes (M1 mot réservé)
BLET	Module LET, incrémente le nombre de clients vérifiant TEST
&COUNT	Compteur du nombre de clients ayant passé plus de 4 mn dans système
TERMINATE (Etiquette TER)	Fin de transaction (le client est supprimé du système)
SIMULATE	Commande de début de la simulation
PUTPIC	« Put picture », commande de stockage des sorties dans le fichier OUT
BUTPIC	Ecrire les sorties dans le fichier OUT
FR(Checkout)/1000	facteur d'utilisation du serveur (ρ)
QM(LINE)	Valeur maximale de la file LINE au cours de la simulation
QT(SYSTIME)	Temps moyen d'attente dans la file SYSTIME
N(TER)	Nombre de clients qui passe le bloc d'étiquette TER
AC1	Horloge (la dernière valeur du temps de simulation)

*/Anomalie du langage : cela ne veut pas dire forcément qu'une file d'attente s forme.

Exemple 1: Simulation d'une file d'attente à un serveur à l'aide de **GPSS/H**. Le temps d'inter-arrivées suit une loi exponentielle de taux $\lambda=1/4.5$; le temps de service S suit une loi normale de moyenne 3.2 et d'écart type $\sigma=0.6$ minutes). On cherche à estimer le temps moyen de répons et la proportion de clients dont le temps de répons est ¥4 minutes.

- **SIMULATE**
- Définir les variables
- INTEGER &LIMIT
  REAL &IAT, &MEAN , &STDEV, & COUNT
  LET &IAT=4.5
  LET &MEAN=3.2
  LET &STDV=.6
  LET &LIMIT=1000
- Ecrire les données dans un fichier

- PUTPIC FILE=OUT, LINES=5 , (&IAT, &MEAN, &STDV, &LIMIT)
- Temps moyen d'inter arrivées **.** minutes
- Temps moyen de service **.** minutes
- Ecart-type du temps de service **.** minutes
- Nombre moyen de clients à servir **.**

*

- **Modules GPSS/H**

GENERATE RVEXPO(1, &IAT) Temps d'inter arrivées, variable de loi exponentielle de moyenne=&IAT

QUEUE SYSTIME Commencer la collecte des données du temps de réponse

QUEUE LINE Le client rejoint la file d'attente

SEIZE CHECKOUT Considérer un client de la file

DEPART LINE Le client en cours quitte la file et commence son service

ADVANCE RVNORM(1, &MEAN, &STDV) Temps de service du client

RELEASE CHECKOUT Le client quitte la zone de service

DEPART SYSTIME Fin de collecte du temps de réponse

TEST GE M1, 4, TER Le temps de réponse est-il supérieur à 4 ?

BLET &COUNT=&COUNT+1 Si oui, ajouter une unité au compteur

TER TERMINATE 1

*

START &LIMIT Simuler pour le nombre requis.

*

- Ecrire les données de sortie dans un fichier
- *
- PUTPIC FILE=OUT, LINES=7, (FR(CHECKOUT)/100, QM(LINE),-
- QY (SYSTIME), &COUNT/N(TER), AC1, N(TER)
- Facteur d'utilisation du serveur ***
- Taille maximale de la file **
- Temps de réponse moyen **.** minutes
- Proportion de clients qui passent 4 minutes
- ou plus dans le système **.** minutes
- Temps de Simulation ***.**
- Nombre de départs ****

END

Résultats de l'expérience.

Temps moyen d'inter arrivées 4.5 minutes

- Temps moyen de service 3.20 minutes
- Ecart-type du temps de service 0.60 minutes
- Nombre moyen de clients à servir 1000
- Facteur d'utilisation du serveur 0.676

- Taille maximale de la file 7
- Temps de réponse moyen 6.33 minutes
- Proportion de clients qui passent 4 minutes
- ou plus dans le système 0.646
- Temps de Simulation 4767.27 minutes
- Nombre de départs 1000

7.3**. SIMAN** (**SI**mulation **M**odelling and **AN**alysis).Ecrit en Fortran puis en C pour les nouvelles versions. Comporte une animation avec **CINEMA,** processeur d'entrée pour ajuster les distributions aux données, processeur de sortie pour obtenir des intervalles de confiance, histogrammes, etc.…Ce langage comporte certains aspects liés à la modélisation des systèmes de production (manufacturiers) : stations, systèmes de transports (convoyeurs, véhicules guidés…L'environnement ARENA, très utilisé dans les milieux universitaires et industriels,

Similitudes entre GPSS/H et SIMAN V

GPSS/H	SIMAN V
GENERATE	CREATE
QUEUE	QUEUE
SEIZE	SEIZE
DEPART	
ADVANCE	DELAY
RELEASE	RELEASE

Le format général d'un Module est

Etiquette (Label) NAME Opérandes : MODIFIERS ; commentaire
(example CREATE,QUEUE,SEIZE,DELAY,RELEASE,DISPOSE).

Exemple 2: Simulation d'une file d'attente à un serveur à l'aide de **SIMAN V**.

```
!Modèle de file d'attente à un serveur
BEGIN
    CREATE :
            IAT :                          ! temps d'inter arrivé
            MARK(Arrtime) ;                attribut de la marque du temps d'inter-arrivée
    QUEUE,
             Line ;                        joindre la file d'attente
    SEIZE :
             Checkout ;                    Occupation du serveur
    DELAY :
              Temps de service ;           Utilisation du serveur
    RELEASE :
              Checkout ;                   Libération du serveur
    TALLY :                                !Evaluation du temps de réponse
              Temps de réponse,    INT(ArrTime) ;
     BRANCH, 1 :                           !Le temps de réponse GE est-il de 4 minutes ?
              IF, (TNOW-Arrtime) >=4, ADD1 :
              ELSE, ADD2 ;
ADD1 COUNT :
                GEFour ;                   Compter les temps de réponse ¥4 minutes
```

```
 ADD2 COUNT :
                Servis ;                    Compter les clients servis
       DISPOSE ;                            Détruire l'entité
```

Résultats de l'expérience.

```
BEGIN :
PROJECT,                Etude, Equipe ;
ATTRIBUTES :            ArrTime
RESOURCES :             Checkout ;
QUEUES :                Line ;
EXPRESSIONS :           1, IAT, EXPO(4.5,1) :          ! interrarrivée
exponentielles
                        2, ServiceTime;   NORM(3.2, 0.6,1) ; Service de loi normale
DSTATS :                NQ(Line) :                     ! Nombre de clients dans
la file
                        NR(Checkout) ;                 Utilisation du serveur
TALLIES :               Response Time ;                Evalue le temps de
réponse
COUNTERS :              GEFour :                       ! temps de réponse >=4
minutes
                        Served, 1000 ;                 Nombre maximum de
clients servis
REPLICATE,              1 ;                            Une réplication
END ;
```

Résultats

Project : Etude — Run execution date : 12/23/1994
Analyst : Equipe — Model revision date : 12/23/1994
Replication ended at time : 4563.48

TALLY VARIABLES

Identifier	Average	Variation	Minimum	Maximum	Observations
Response Time	7.6973	0.78156	1.6324	37.012	1000

DISCRETE-CHANGE VARIABLES

Identifier	Average	Variation	Minimum	Maximum	Final value
NQ(Line)	0.97916	1.7274	.00000	11.000	1.0000
NR(Checkout)	.70792	.6423	.00000	1.0000	1.0000

7.4. SIMSCRIPT II.5 Permet la construction de modèles orientés processus ou orienté événement. Doté d'un animateur graphique SIMGRAPHICS ; permet de produire des histogrammes et graphes.(**http://www.hyperdictionary.com/dictionary/SIMSCRIPT**).

Exemple : Modèle de file d'attente à un serveur.
Preamble

```
    Processes include CUSTOMER, CUSTOMER.GENERATOR
    Every CUSTOMER.GENERATOR has
    a CUST.MEAN.IAT
    Define CUST.MEAN.IAT as a real variable
    Ressources includes SERVER
    Define NUMBER OF CUSTOMERS
            MEAN.INTER.ARRIVAL.TIME,
            MEAN.SERVICE.TIME,
            SERVICE.STD.DEV as real variables
Define TIME.IN.SYSTEM, FOUR.OR.MORE as real variables
Define CUSTOMERS.SERVED as an integer variable
Tally MEAN.TIME.IN.SYSTEM as the average of TIME.IN.SYSTEM
Tally MAX.QUEUE as the max of N.Q.SERVER
Accumulate UTILIZATION as the average of N.X.SERVER
Define minutes to mean units
End
Main
   Write as « SIMSCRIPT II.5 Modèle de file d'attente»,/
    Let NUMBER.OF.CUSTOMERS=1000
    Let MEAN.INTERARRIVAL.TIME=4.5
    Let MEAN.SERVICE.TIME=3.2
    Let SERVICE.STD.DEV=0.6
    Create every SERVER(1)
    Let U.SERVER (1)=1
    Create a CUSTOMER.GENERATOR
    Let CUST.MEAN.IAT(CUSTOMER.GE NERATOR)=MEAN.INTER.ARRIVAL.TIME
Activate this CUSTOMER.GENERATOR now
Start Simulation
Call REPORT.RESULTS
End

PROCESS CUSTOMER
  Define ARRIVAL.TIME as a real variable
  Let ARRIVAL.TIME=TIME.V
  Request 1 SERVER(1)
  Work normal.f(MEAN.SERVICE.TIME, SERVICE.STDV.DEV,2) minutes
  Relinquish 1 SERVER(1)
  Let TIME.IN.SYSTEM=TIME.V-ARRIVAL.TIME
  IF  TIME.IN.SYSTEM not less than 4.0
  Add 1 TO FOUR.OR.MORE
  Endif
End
Process CUSTOMER GENERATOR
  While CUSTOMERS.SERVED less that NUMBER.OF.CUSTOMERS
Do
          Wait exponential.f(CUST.MEAN.IAT(CUSTOMER.GENERATOR),1)
Minutes
          Create a CUSTOMER
         Add 1 to CUSTOMERS.SERVED
         Activate this CUSTOMER now
```

```
   Loop
End

Routine REPORT RESULTS
    Print 4 lines with MEAN.INTER.ARRIVAL.TIME, MEAN.SERVICE.TIME,
        SERVICE.STD.DEV, NUMBER.OF.CUSTOMERS
       thus
              Mean interarrival time                          **.**
              Mean service  time                              **.**
              Standard deviation of service time              **.**
              Number of customers to serve                    **.**

Skip 2 output lines

Print 7 lines with UTILIZATION(1), MAX.QUEUE(1),
MEAN.TIME.IN.SYSTEM,
         (FOUR.OR.MORE/NUMBER.OF.CUSTOMERS), TIME.V,
CUSTOMERS.SERVED
        thus
              Server utilization                              **.**
              Maximum line length                            *****
              Average response time                           **.**
              Proportion who spent four
                 minutes or more in system                    **.**
              Simulation run length                         *****.**
              Number of customers                           ******
  End
```

Résultats de l'expérience pour le modèle d'attente.
SIMSCRIPT II.5 Modèle de file d'attente.

Mean interarrival time	4.50
Mean service time	3.20
Standard deviation of service time	0.60
Number of customers to serve	1000
Server utilization	0.71
Maximum line length	5
Average response time	6.47 minutes
Proportion who spent four minutes or more in system	0.68
Simulation run length	4500.48 minutes
Number of customers	1000

7.5. SLAMSYSTEM : SLAM : permet un ordonnancement d'évènement ou une orientation d'interaction de processus ; SLAMSYSTEM permet de construire, d'animer et d'exécuter les modèles de simulation en SLAM.
Une branche en SLAM représente l'exécution du temps (représente donc une activité. codée comme état ACTIVITY Les nœuds sont utilisés pour représenter l'arrivée d'un événement, retard ou attente conditionnelle (nœud QUEUE), évènement de départ (nœud TERMINATE

Exemple : modèle de file d'attente à un serveur en SLAM II

```
LIMITS, 1,0,30 ; MODEL CAN USE 1 FILE, MAX NO. OF SIMULTANEOUS ENTRIES
30
NETWORK ;       BEGINNING OF MODEL
CREATE, EXPON(4.5) ;                    CUSTOMERS ARRIVE AT
CHECKOUT
 QUEUE(1) ;                             CUSTOMERS WAIT FOR SERVICE IN
                                            QUEUE FILE ONE (1)
ACTIVITY(1)/1, RNORM(3.2,.6) ;          CHECKOUT SERVICE TIME IS N(3.2,
0.6)
TERMINATE, 1000 ;                       SIMULATE UNTIL 1000 CUSTOMERS
ARE CHECKED OUT
ENDNETWORK ;                            END OF MODEL
                    END OF SIMULATION
```

Résultats de l'expérimentation à l'aide de SLAM II.

```
SLAM II SUMMARY REPORT
SIMULATION PROJECT SINGLE SERVER QUEUE       BY BANKS CARLSON
NELSON
DATE 1/20/1995                                RUN NUMBER   1 OF 1
CURRENT TIME    .4134E+04
STATISTICAL ARRAY CLEARED AT TIME  0.

                                **FILE STATISTICS**

FILE          ASSOCIATED   AVERAGE   STANDARD   MAXIMUM   CURRENT
AVERAGE
NUMBER    NODE TYPE     LENGTH     DEVIATION   LENGTH      LENGTH
WAITING  TIME

   1             QUEUE          1.221          1.4672              8             0
4.6677
   2                            1.7694           .4212              3             2
2.6523

                            **SERVICE ACTIVITY STATISTICS**

ACTIVITY   START NODE   SERVER      AVERAGE     STANDARD  CURRENT
AVERAGE     MAX IDLE  MAX BUSY  ENTITY
INDEX       LABEL/         CAPAC       UTILIZ       DEVIATION   UTILIZAT
BLOCKAGE   TIME/        TIME/       COUNT
               TYPE                                                  SERVERS
SERVERS

   1           QUEUE          1             .7694          .4212             1
0.0000         21.4036      232.2203       1000
```

7.6. MODSIM III (orienté objet). Syntaxe et structure basée sur Modula-2, permet une interface avec C de telle sorte qu'on puisse utiliser les bibliothèques source de C ; on peut

inclure un code objet ; une interface avec l'animateur CACI de SIMGRAPHICS. Les versions récentes nécéssitent C++.

Exemple : modèle d'attente à un serveur à l'aide de MODSIMIII
Il existe un code pour : 1.Initialiser le programme et c réer les objets participants ; 2.staitisytqqiues ; 3. report des résultats.
Le programme débute en important quelques objets et procédures nécessaires à partir des bibliothèques de MODSIMIII

```
From SimMod   IMPORT   StartSimulation, SimTime
From RandMod  IMPORT RandomObj ;
From ResMod IMPORT ResourceObj ;
From StatMod IMPORT RstatObj ;
```

Déclaration de CustomerGeneratorObj :

```
CustomerGeneratorObj=
    OBJECT
        numCustomers : INTEGER ;
       TELL, METHOD GenCustomers(IN numToGen : INTEGER) ;
END OBJECT ;
```

Déclaration pour MyMonitorObj (pour les statistiques)

```
MyMonitorObj=
    MONITOR REAL OBJECT(RSTATObj) ;
         totCustGE4 : INTEGER ;
        OVERRIDE
            LMONITOR METHOD access ;
END OBJECT ;
```

Déclaration des variables utilisées

```
VAR
     server                    :   ResourceObj ;
     custGenerator             :  CustomerGeneratorObj ;
     svcRanGen                 : RandomObj ;
     meanServiceTime,
     serviceStdDev             : Real ;
     numCustToGen              : INTEGER ;
    timeInSystem               : LMONITORED REAL BY MyMonitorObj ;
```

Code d'implémentation des méthodes pour chaque objet.

```
TELL METHOD BeServed ;
VAR
      arriveTime     :  Real ;
      serviceTime    : Real ;
BEGIN
      arriveTime :=SimTime() ;
      WAIT FOR  server TO Give(SELF,1) ;
           serviceTime :=svcRangen.Normal(meanServiceTime,
serviceStdDev) ;
           WAIT DURATION serviceTime ;
           END WAIT ;
           timeInSystem :=SimTime()-arriveTime ;
```

```
        DISPOSE(SELKF) ;
    END METHOD ;
```

Programme principal :

```
BEGIN (   main  )
    OUTPUT (« MODSIM  III model of single server queue ») ;
    meanServiceTime :=3.2 ; serviceStdDev :=0.6 ;
    numCustToGen :=1000 ;
    NEW(svcRanGen) ;
    NEW(server) ;
    ASK server TO Create(1) ;
   ASK server TO SetAllocationStats(TRUE) ;
   ASK server TO SetPendStats(TRUE) ;
   NEW custGenerator TO GenCustomers(numCustToGen) ;
   StartSimulation ;
   ReportResults ;
 END MODULE.
```

SimTime	Fonction qui retourne le temps de simulation en cours
StartSimulation	procédure qui lance la simulation
RandomObj	Objet réalisant les générateurs de diverses lois de probabilité
ResourceObj	modélise le serveur
RStatObj	construit un monitor statistique appelé MyMonitorObj
svcRanGen	Objet : générateur de nombres aléatoires de loi normale
timeInSystem	variable de type LMONITORED REAL BY MyMOnitorObj
numCustomers	champ qui garde trace du nombre de clients générés
GenCustomers	méthode qui utilise numToGen comme paramètre

7.7. QNAP (Queuing Network Analysis Package). Développé par INRIA & Bull(1980), commercialisé par Simulog. Langage interprété spécialisé files d'attente développé en FORTRAN.C'est un langage algorithmique proche du Pascal qui comporte un éditeur graphique (Modline). Il permet la résolution de modèles avec un moteur de simulation à évènements discrets, mais utilise également des techniques analytiques exactes ou approchées (décomposition en forme produit, techniques numériques de résolution de systèmes d'équation algébriques, techniques de diffusion… http://www.simulog.com

Structure d'un programme QNAP2 et conventions syntaxiques

Une commande se termine par : BEGIN.END pour plusieurs instructions dans un même bloc.
Les mots-clés doivent être en majuscule (langage case-sensitive)
Une ligne de commentaires commence par &
L'opérateur d'affectation est :=
L'opérateur testant l'égalité est =

Modules/Attributs	Description
/CONTROL/	Paramètres généraux du modèle (ex : temps de simulation TMAX=x ;

/DECLARE/	Déclaration des objets (identificateurs) et variables utilisés, initialisation des variables(-types INTEGER, REAL, BOOLEAN,STRING, QUEUE, CUSTOMER ; -tableaux
/EXEC/	Section algorithmique incluant le lancement de la simulation ou de l'analyse
/STATION/	Description d'une file d'attente (service, politique, routage,…
CUSTOMER	Type de chaque client du système ; attributs descriptifs par défauts (CPRIOR, CQUEUE, CCLASS, FATHER, NEXT, PREVIOUS,…Enrichissement : /DECLARE/ CUSTOMER INT taille ;
SERVICE	Résultat issu de la trace standard : temps moyen de service
BUSY PCT	Idem : taux d'activité du serveur
CUST NB	Idem : moyen de clients dans la station
RESPONSE	Idem : temps moyen de réponse
SERV NB	Idem : nombre total de clients servis

Avantages de QNAP2: bien adapté pour modéliser rapidement et simuler des réseaux de files d'attente complexes; présence de solveurs analytiques.

Inconvénients : Nécessite plus de développement ; l'interface graphique est obsolète ; pas d'environnement de programmation.

Outils similaires : **QNA** (Queueing Network Analyzer), développé par les « Bell Laboratory, USA). Ce logiciel évite les "solveurs" analytiques traditionnels (méthodes numériques de résolution d'équations algébriques ou différentielles). Il fait appel plutôt à la théorie des inégalités et encadrement par bornes, techniques heuristiques. Les évaluations sont moins précises, mais plus flexibles et surtout plus rapides.

Exemple 1 : /DECLARE/ INTEGER a=1 ;
QUEUE foo, toto;
REF QUEUE ptr;
BOOLEAN tab(5);
REAL b;
CUSTOMER INT nbpassage;

Exemple 2: Type QUEUE : manipulation

```
PROCEDURE Liste(F) ;
REF QUEUE F ;
REF CUSTOMER Client ;
INTEGER Numero;
BEGIN
    WRITELN("*********liste clients**********");
    CLIENT    := F.FIRST; Numero  := 1;
    WHILE  (CLIENT<>NIL)  DO
    BEGIN
       WRITE( "lient  #", Numero,  "avec  ");
       WRITELN(Client.NbPassages, "par la CPU");
       Client   := Client.NEXT ;  Numero := Numero +1 ;
     END;
```

```
      WRITELN("**********************************");
   END;
```

Exemple 3: /STATION/: déclaration d'une file

```
NAME=nom ;
[optionnel] CAPACITY =x ;
[optionnel] REJECT=destination ;
[optionnel] TYPE=[SINGLE|SOURCE|MULTIPLE(n)|INFINITE] ;
SERVICE=[EXP(t)|UNIFORM(x,y)|…] ;
[optionnel] SCHED=[FIFO|LIFO|PRIOR|PREEMPT]
[optionnel] INIT=x;
TRANSIT=destination, probabilité ;
```

Exemple 4 : Exemple d'une station source

```
/STATION/ NAME =source ;
      TYPE =SOURCE.
      SERVICE =CST(10.) ;
      TRANSIT =toto, 0.5, foo , 0.3, OUT;
```

Exemple 5: station serveur

```
/STATION/ NAME=foo;
   CAPACITY =10;
   REJECT=TRANSIT(OUT);
    SERVICE =BEGIN
                  IFCUSTOMER.nbPassage=1 CST(10)
                  ELSE CST(20);
                  CUSTOMER.nbPassage:= CUSTOMER.nbPassage+²1;
                  END;
  TRANSIT =toto, 0.5, foo, 0.3, OUT;
```

Exemple 6: Lancement effectif de la simulation

```
/EXEC/
      BEGIN
        FOR j :=1 STEP  1  UNTIL  10  DO
        BEGIN
            WRITELN ("Début simulation", j);
            SIMUL; & ou SOLVE ou Markov
            WRITELN(« Fin de simulation »,j) ;
          END ;
         END ;
```

Exemple 7 : Exemple de trace. (voir annexe).
Exemple 1: Simulation d'une file d'attente à un serveur à l'aide de QNAP2

7.8. SIMULA . Développé au centre de calcul norvégien d'Oslo NCC entre 1962 et 1967 sur la base des idées d'ALGOL 60 (http://engin.umd.edu/cis/course.des/cis400/Simula/Simula.html).

7.9. SSJ (Stochastic Simulation in Java). Bibliothèque (librairie) de classes dans le langage de programmation Java qui offre des facilités de programmation pour la simulation. Conçu initialement pour la simulation stochastique à évènements discrets, mais peut être adapté à la simulation continue (équation différentielle). Certaines idées de SSJ découlent des paquets de programmes DEMOS (basé sur le langage SIMULA, SIMPascal (basé sur langage Pascal) et SIMOD (basé sur le langage Modula-2). Développé au département d'Informatique et Recherche opérationnelle de l'Université de Montréal. (http://www.iro.umontréal.ca/~simardr/ssj).

SSJ est implémenté comme une collection de classes dans le langage Java. Les classes prédéfinies contiennent des facilités pour générer des nombres aléatoires de différentes distributions, collecter des statistiques, gérer l'horloge et l'échéancier, la liste des évènements futurs, synchroniser les interactions entre les processus concurrents simulés etc…SSJ supporte le point de vue évènements et le point de vue processus, mais aussi la simulation continue (où certaines variables évoluent selon des équations différentielles). Les 3 points de vue peuvent être combinés.

Classes et packages	Description
RandomStream	Interface qui permet de générer des nombres aléatoires ; RandMrg (uniforme) ; autres Rand1, Rand2, ….
StateProbe	Classe pour récolte de statistiques et calcul d'intervalles de confiance
Tally	Sous classe de StateProbe
Accumulate	Sous classe de StateProbe
List	Classe qui implémente des listes doublement chaînées, avec des outils d'insertion, de déplacement et de visualisation des objets dans la liste, une récolte automatique des statistiques.
Object	Différents types contenus dans List
SIM	Classe qui gère la simulation à évènements discrets, contient une horloge de simulation et un moniteur central
EventList	Interface qui fournit différentes implémentations d'évènements list
Sim.init	Permet de modifier l'implémentation de l'événement list
Sim.start	Enclenche la simulation en avançant l'horloge à l'instant du premier événement dans la liste d'évènements, élimine cet événement de la liste, et l'exécute. Ceci est répété jusqu'à ce qu'on appelle Sim.stop ou lorsque la liste se vide.
Sim.time	Retourne le temps courant de l'horloge de simulation
Event, Process	Permettent de créer et ordonnancer les évènements et processus
Continuous	Outil pour la simulation continue
Resource	Mécanisme pour la synchronisation : correspond à une ressource à capacité limitée et file d'attente
Bin	Relation client/serveur entre processus ; consiste à une pile de

	jetons libres et une file de processus attendant les jetons
Condition	Processus qui attend qu'une certaine condition booléenne soit vraie pour pouvoir continuer son exécution

Exemple : Simulation d'une file d'attente M/M/1.
Vous trouverez ici trois manières de simuler une telle file à l'aide de SSJ :

(i) programme orienté événement (annexe 2)
(ii) programme orienté processus (annexe 3).

Classe ou sous-classe	description
Event	Classe événement
Arrival	Arrivée d'un client
Departure	Départ d'un client
EndOfSim	Fin de la simulation
QueueEv	Crée deux suites de nombres aléatoires (listes) et deux collecteurs de statistiques
genArr, genServ	Génère des nombres artificiels :temps entre arrivées et temps de service
waitList, servList	Listes contenant les clients en attente et en service
CustWaits (de classe Tally)	Récolte les statistiques des temps d'attente
update	Introduit une nouvelle observation à chaque début de service
totWait (de classe Accumulate)	Calcule l'intégrale (et éventuellement la moyenne temporelle) d'un processus aléatoire à temps continu à trajectoire en escalier (ici c'est la longueur de la file en fonction du temps)
totWait.update	Appeler dès qu'il y a changement dans la file, met à jour le temps total d'attente de tous les clients depuis le début de la simulation
Chaque client est un objet à deux champs (arrivTime, servTime	Mémorisent temps d'arrivée et de service de ce client
QueueEv	Initialise l'horloge et la liste d'évènements
Sim.start	
1000	Fin de la simulation
Rand1.expon(g,m)	Retourne un nombre aléatoire de loi exponentielle de moyenne m, généré par le flux aléatoire g
actions	Décrit ce qui se passe lorsque lors d'une arrivée

(i) **Programme de simulation orienté événement d'une file M/M/1**.

```
public class QueueEv  {

    static   final  double meanArr       =10.0;
    static   final  double meanServ      =9.0;
```

```
 static    final  double timeHorizon   =1000.0;

 RandMrg  genArr       =new  RanMrg  ();
 RandMrg  genServ      =new  RanMrg  ();
 List          waitList     =new  List    (''Customers waiting in the queue'');
 List          servList     =new  List    (''Customers in service'');
 Tally         custWaits    =new  Tally   (''Waiting times'');
 Accumulate    totWait  =new  Accumulate  (''Size of Queue'');

 class  Customer  {  double arrivTime,  servTime;  }

 public static void main  (String[]  args)  {  new  QueueEv();  }

public  QueueEv()  {
     Sim.init();
     new  EndOfSim().schedule  (timeHorizon);
     new   Arrival().schedule (Rand1.expon (genArr, meanArr));
     Sim.start ();
     }
  class  Arrival extends  Event  {
      public  void  actions()  {
           new  Arrival().schedule  (Rand1.expon  (genArr, meanArr));
                                                  //  The next arrival.
           Customer  cust    =new Customer ();   // Cust just arrived.
           cust.arrivTime=Sim.time();
           cust.servTime=Rand1.expon (genServ, meanServ);
           if  (servList.size()  >0  {            // must join the queue.
              waitList.insert  (cust, List.LAST);
              totWait.update  (waitList.size());
           }   else  {
                 servList.insert  (cust, List.LAST);
                 new  Departure().schedule  (cust.servTime);
                 custWaits.update  (0.0);
                 }
            }
     }
  class Departure extends Event  {
      public void actions  ()  {
            servList.remove  (List.FIRST);
            if  (waitList.size ()>0)  {
                 //  Starts service for next one in queue.
                Customer cust =(Customer) waitList.remove (List.FIRST);
                servList.insert  (cust, List.LAST);
                new  Departure ().schedule (cust.servTime);
                custWaits.update  (Sim.time ()-cust.arrivTime);
                totWait.update (waitList.size ()) ;
                }
            }
     }
  class EndOfSim extends Event  {
```

```
        public  void actions  ()  {
              custWaits.report() ;  totWait.report();
              Sim.stop();
                    }
                 }
           }
```

Résultats

```
REPORT   on Tally  stat.  Collector  ➔  Waiting times
     Min              max            average            standard  dev
nb.obs.
      0              113.721          49.554             22.336                  97

report   ON Accumulate  stat.  Collector  ➔   Size of queue
```

(ii) Ici, une approche orientée processus, comme en contienne de nombreux langages de simulation, en particulier SSJ.

(iii)

```
public class QueueProc  {

      static    final  double meanArr        =10.0;
      static    final  double meanServ       =9.0;
      static    final  double timeHorizon    =1000.0;

      Resource  server        =new  Resource (1,''server'');
      RandMrg  genArr         =new  RanMrg  ();
      RandMrg  genServ        =new  RanMrg  ();

     public static void main  (String[]  args)  {  new  QueueProc();  }

     public  QueueProc()  {
           Sim.init();
           Server.collectStat  (true);
           new  EndOfSim().schedule  (timeHorizon);
           new   Customer().schedule (Rand1.expon (genArr, meanArr));
           Sim.start ();
           }
      class  Customer extends  Process  {
           public  void  actions()  {
                 new  Customer().schedule  (Rand1.expon  (genArr, meanArr));
                 server.request  (1);
                 delay (Rand1.expon (genServ, meanServ));
```

```
            server.release  (1);
            }
      }
class EndOfSim extends Event  {
      public  void actions  ()  {
            server.report();
            Sim.stop();
                  }
               }
      }
```

```
REPORT   ON RESOURCE   : Server
       From time :    0.0    to time   :   1000.0
                          Min              max          average
Std.Dev.        nb.    Obs.
Capacity:                  1                1             1
Utilisation                0                1           0.999
Queue  Size                0               12           4.85
Wait                       0           113.721           49.554
```

7.10.MOSEL (Modeling , Specification and Evaluation Language)

Environnement de modélisation et simulation développé essentiellement à l'université d'Erlangen (Allemagne) orienté vers l'évaluation des performances, en particulier de fiabilité. http://www4.informatik.uni-erlangen.de/Projects/MOSEL/. Il peut être utilisé aussi bien pour la modélisation de systèmes de production que les systèmes informatiques (y compris réseaux mobiles).

Utilise divers formalismes de modélisation :

- Modèles de systèmes et réseaux de files d'attente
- Modèles de Réseaux de Petri
- Modèles de graphes de précédence
- Arbre des fautes
- Modèles de Markov
- Réseaux d'activité stochastique
- Algèbres de processus stochastique

Il permet de générer automatiquement l'espace des états du modèle dynamique, d'effectuer une transformation en un processus aléatoire ou un modèle de simulation à évènements discrets, d'évaluer les probabilités d'état numériquement ou par simulation, et enfin de déduire les mesures de performance. L'environnement MOSEL est capable de transcrire les modèles dans une description en termes de Réseaux de Petri Stochastiques Généralisés pour la bibliothèque SPNP. Le modèle sémantique est transformé en un processus stochastique qui est numériquement résolu à l'aide d'algorithmes appropriés et les mesures de performance et de fiabilité spécifiées par l'utilisateur sont générées dans un fichier de résultats. Les représentations graphiques peuvent être également obtenues à la demande.

Un modèle MOSEL comporte 5 composantes :

1. **Déclaration des paramètres.**

```
CONST  pi =3.14159265358979.
PARAMETER lambda =0.25, 0.30 , 0.35 ;
ENUM cpu_states = {idle, user, kernel, driver} ;
```

2. **Définition des composantes**. La partie NODE permet de spécifier les composantes (noeuds) du modèle. Chaque nœud comporte un nombre entier de jobs (ou de jetons) (inférieur à la capacité). Pour chaque composante, il faut donner le nom et la capacité.

```
NODE  N1[k] =k;                 /* range:  0..k  */
NODE  cpu[cpu_states];          /*  range:  0..3 */
```

La partie ASSERT est utilisée pour spécifier les états prohibés du système. Souvent cette assertion est utilisée pour exprimer l'isolation du système modélisé. Pour un réseau fermé, le nombre total de marques dans les noeuds du modèle est toujours constant :

3. **Définition des transitions**.

Les transitions (appelées règles dans MOSEL) déterminant le comportement du modèle sont spécifiées dans cette partie. Chaque règle comporte une partie globale, une partie optionnelle et une partie locale :

```
FROM node_1  TO  node_2  WITH  mue_1 ;

IF  (cpu_control  ==idle)  FROM node_1  WITH  mue_1
     THEN  (  TO node_2  WEIGHT  p_12 ;
              TO node_3  WEIGHT  p_13 ;  )
```

La première règle consiste en une seule partie locale et spécifie une transition de node_1 à node_2 avec un taux mue_1. Le mot-clé WITH indique que la transition est exponentiellement temporisée. La seconde instruction montre comment une règle contenant une partie locale peut être utilisée pour un branchement probabiliste. La transition temporisée mue_1 provenant de node_1 conduit à node_2 avec une probabilité p_12 ou à node_3 avec une probabilité p_13 (ces poids doivent être déclarés au préalable). Chaque branche est ainsi sélectionnée selon la probabilité correspondante et la transition est immédiate. La condition IF (cpu_control == idle) dans la partie globale de la règle est utilisée pour la synchronisation. Le client (le job) ne peut quitter node_1 que si le nœud cpu_control est à l'état libre.

4. **Résultats** : L'utilisateur trouve en sortie les mesures de performance et de fiabilité qu'il aura spécifié au préalable telles que les valeurs moyennes MEAN, le taux d'utilisation UTIL et la distribution du nombre de clients DIST pour un nœud spécifié :

```
RESULT  N_1 =MEAN (node_1) ;
RESULT  rho_1 = UTIL (node_1);
RESULT DIST node_1;
```

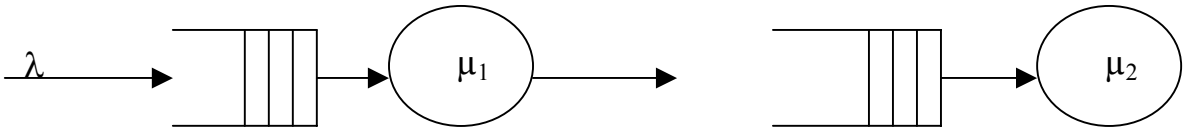
On peut évaluer des probabilités d'évènements:

```
RESULT p_working = PROB (working >0);
RESULT  rho_cpu_kernel = PROB (cpu == kernel
                                    AND  cpu_state == up);
```

5. **Partie graphique**: L'utilisateur peut dans cette partie optionnelle représenter graphiquement les mesures de performance en fonction des paramètres spécifiés. Si on souhaite représenter la taille moyenne de la file d'attente en fonction du taux d'arrivée lambda :

```
PICTURE  « Mean Queue Length »
    PARAMETER   lambda
   CURVE   mean_queue_length
```

Exemple1: **Réseau de files d'attente ouvert en tandem**.



Chaque nœud consiste en un système M/M/1 avec file d'attente FIFO. Le programme MOSEL est alors le suivant :

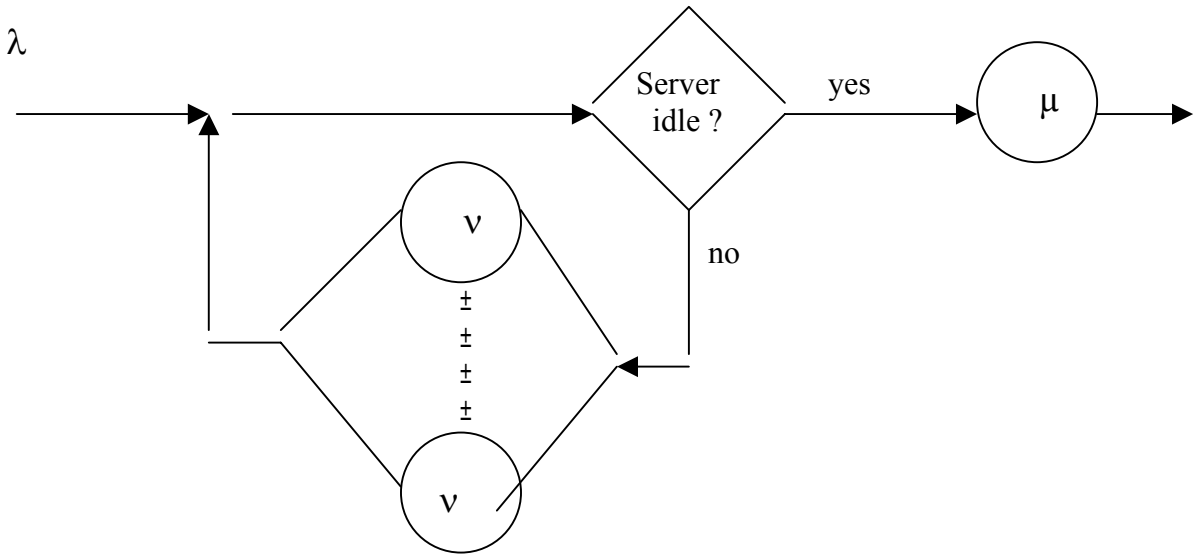
```
1   //  Tandem network
2
3    // Parameter declaration part
4
5   PARAMETER  K := 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ;
6   CONST lambda :=0.25 ;
7   CONST mue_1 :=0.28 ;
8   CONST mue_2 :=0.22 ;
9
10   // System component part
11
12  NODE  N1 (K) = 0 ;
13  NODE N2 (K) =0 ;
14  NODE num (K) ;
15
16    //Transition part
17
18  FROM EXTERN TO N1, num  WITH lambda;
19  FROM  N1 TO  N2  WITH mue1;
20  FROM  N2 , num TO EXTERN  WITH mue2;
21
22  // Result part
23
```

```
24  PRINT rho1 =UTIL (N1);
25  PRINT rho2 = UTIL (N2);
26  PRINT throughout =rho2 * mue2;
27  PRINT  WIP  = MEAN (num);
28
29   // Picture part
30
31  PICTURE “utilization”
32  PARAMETER  K
33  CURVE  rho1;
34  CURVE rho2;
```

Les paramètres constants lambda, mue1 et mue2 sont déclarés au préalable. L’analyse est faite en supposant que le nombre maximum de clients dans le système K varie de 1 à 10.

Exemple 2 : Systèmes de file d’attente avec rappels.

Considérons le modèle de file d’attente avec répétition d’appel le plus simple M/M1 (Retrial queue dans la littérature anglo-saxonne). Le client qui trouve à son arrivée le serveur occupé quitte le système temporairement, mais réitère son appel à des instants aléatoires jusqu’à ce que le canal se libère et puisse le prendre en charge.



Le modèle sur le schéma représente un modèle à plusieurs serveurs : μ est le taux de service de chaque serveur, ν le taux de rappel d’un client qui se trouve en orbite (en état de répétition d’appel). Les lois d’inter-arrivées, de service et de rappel sont exponentielles. Dans notre cas, un seul serveur. Pour l’analyse de ce modèle en MOSEL, on peut introduire un nœud fictif node D représentant les appels rejetés et en instance de rappels qui est de type M/M/ ¶ -IS. En d’autres termes, il n’y a pas de file d’attente et le taux de service dans ce nœud est le taux de rappel.

```
//              M/M/1_Retrial System

//              Declaration part
```

```
PARAMETER  lambda  := 0.01, 1, 2, 5, 8, 9, 9.5, 9.9;
PARAMETER nue  := 0.01, 0.1, 1, 1000000;
CONST  mue  := 10;
ENUM  status  := {idle, busy };
CONST  N := 100;      // Maximum number of requests

//                Component definition part
NODE  server[status] =idle; //server initially idle
NODE  D[n-1];  // delay node contains maximal N-1 req.
NODE num[N]; // counts total number of requests

//                Transition part

     FROM  EXTERN  TO D, num  WITH  lambda;
     FROM  D  TO D, num  WITH D*nue;
     }
IF  (server  == busy) {
     FROM EXTERN  TO D, num WITH lambda;
     FROM  d  to  d  with D*nue;
     FROM  server , num  TO EXTERN  WITH  mue ;
     }

//                              Result part
PRINT  rho  =  UTIL (server);   //  Server utilization
PRINT  K = MEAN  (num);      // Mean number of request
PRINT  Q = MEAN (D);          // Mean queue length
PRINT  T = K/lambda;            //  Mean system time

//                          Picture part
PICTURE  “Mean Queue Length”
Parameter   LAMBDA
CURVE  Q;

PICTURE  “Utilization”
PARAMETER   lambda
CURVE  rho;
```

Stationary analysis of “mm1.retrial.msl” by SPNP

Parameters: lambda = 0.01 nue =0.01 Results: rho = 0.001 K= 0.002002 Q= 0.001002 T = 0.2002	Parameters: lambda = 9.9 nue =1 Results: rho = 0.908255 K= 90.806 Q= 89.8977 T = 9.17232
Parameters: lambda = 0.01	Parameters: lambda = 9.9

nue =0.1	nue =1e+06
Results:	Results:
rho = 0.001	rho = 0.984317
K= 0.0011011	K= 41.6014
Q= 0.000101101	Q= 40.617
T = 0.11011	T = 4.20216

Remarques :

(i) Le taux de rappels est déclaré comme paramètre, car pour ce type de modèle on est intéressé en général par l'étude de l'influence des rappels sur les mesures de performance.

(ii) Nous reportons ici l'analyse faite par l'exécution de SPNP.

(iii) Le modèle M/M/1 est à capacité infinie, alors que le processus aléatoire du modèle SPNP comporte un nombre fini d'états. Ce qui explique le fait que le programme MOSEL fixe un nombre maximal fini de clients N(variant de 0 à 100). Les résultats obtenus ne concordent donc pas tout à fait avec les résultats exacts de l'exercice chapitre 5 qui concernent un système à capacité infinie.

(iv) L'environnement MOSEL grace à sa partie graphique génère les graphes demandés à l'aide de l'instruction CURVE.

7.11.Simulateurs et langages dédiés aux systèmes manufacturiers

7.11.1.SIMFACTORY II.5. (Factory simulator). Ecrit en SIMSCRIPT II.5 et MODSIM III pour ingénieurs qui ne font pas de la simulation à temps plein. (PC environnement Windows ainsi que diverses stations).Le modèle est défini par étapes : agencement des stations de traitement, buffers, moyens de transport, les produits, ressources. Animation à base d'icônes et de palettes. L'affichage des résultats fournit l'utilisation des équipements, les débits, utilisation des buffers, statistiques (moyennes, écart-types, intervalles de confiance).

7.11.2.ProModel. Environnement Windows, applications à 32-bits, permet d'ajouter des routines en Pascal ou en C.

7.11.3.AutoMod. (AutoSimulations, Inc.).(http://www.autosim.com). Combine les aspects de GPSS et celle d'un simulateur spécifique (simulateur d'atelier ou job shop), graphisme 3D, paquets de programmes statistiques (AutoStat), ordonnancement (AutoSched). Fonctionne sur PC ou Stations UNIX.

7.11.4.Taylor. II. Appelé aussi Showflow. (produit allemand développé par F&H Simulations B.V.). (http://www.showflow.com) Entités : éléments, jobs, routages et produits. Eléments types : in-out, machine, buffer, conveyor, transport, chemin, stock, réservoir. Opérations de base : processing, transport, stockage. Utilise un macro langage appelé TLI(Taylor Langage Interface) interface avec Basic, Pascal et C. Animation 2D et 3D.

7.11.5.WITNESS. (AT & T Istel).

7.11.6.AIM (Analyzer for Improving Manufacturing) (P c'est un composant de simulation de FACTOR. autres composants : Factor Production Manager et FI-2. (disponible sur plate-forme OS/2. construit autour d'une base de données relationnelle qui stocke les données et les résultats de simulation.

7.11.7.Arena (System Modeling Corporation). Développé à partir de SIMAN. L'utilisateur n'a pas besoin de connaître SIMAN **.** (http:/www.arenasimulation.com) . Il semble être utilisé surtout dans les domaines des banques et assurances et semble très simple à apprendre. Il comprend un module pour les ajustements de diverses lois statistiques. Il ne considère toutefois qu'un nombre limité de discipline de service : FIFO, LIFO, highvaluefirst, LowValueFirst.

7.12.Simulateurs et langages dédiés réseaux télécommunication et mobiles.

Les langages décrits ci-dessus peuvent en principe être utilisés pour la simulation de réseaux mobiles. Un exemple est donné dans [10] qui suggère l'utilisation de M/M/m/m à l'aide de l'environnement MOSEL. Il existe toutefois d'autres langages spécialisés.

7.12.1.Glomosim (Global Mobile system Simulator) (http://pcl.cs.ucla.edu). Développé comme une bibliothèque de simulation séquentielle et parallèle orientée réseaux sans fils. Il est conçu comme un ensemble de modules pouvant chacun simuler un protocole de communication sans fil particulier. Développé à base de PARSEC (PARallel Simulation Environment for Complex Systems), un langage de simulation parallèle basé sur C, utilisant l'approche basée message à la simulation à évènements discrets.

7.12.2.Network simulator Http://www.isi.edu/nsnam/ns/ns-documentation.html
Développé essentiellement à l'université de Berkeley et adapté à la simulation de réseaux . Il concerne un ensemble de protocoles internet y compris les réseaux terrestres, sans files (wireless) et satellites. Il comprend un éditeur graphique et un module d'animation (nam : Network Animator). Il permet de simuler des réseaux de différentes topologies selon divers algorithmes de routage, générer différentes sources de trafic (web, ftp, telnet, cbr, trafic stochastique, diverses disciplines de files d'attente. Il visualise alors les flux de paquets, l'évolution des protocoles.

7.13.Bibliographie sur les langages de simulation

1.Banks J., Carlson J.S., Nelson B., Discrete System Simulation , second ed., Prentice Hall,New Jersey, 1996. (les exemples sont en majorité tirés de ce livre).
2.Gordon G., The application of GPSS V to discrete system simulation , Prentice Hall , Englewood Cliffs , NJ, 1975.
3.IBM Corp., GPSS V User's manual , form N° SH 20-0851, 1970.
4.Dimsadel B., Markowitz H.M., A description of the SIMSCRIPT Langage , IBM systems J., 3,1,57-67, 1964.
5.Markowitz H.M., Hausner B., Karr H.W., SIMSCRIPT -A simulation langage , Prentice Hall , Englewood Cliffs , NJ, 19+63.
6.Palme J., A comparison between SIMULA and FORTRAN , BIT, 8,3, 203-209, 1968.

7.Knuth D.E., Mc Neley J., SOL: A symbolic langage for General Purpose System Simulation , IEEE on Electronic Computers , EC-13, 4, 401-414, 1963.
8. IBM Corp SIMPL/1 User's manual, NY, 1972.
9. Manuel de documentation QNAP2
10. Pierre L'écuyer, A Java Library for Stochastic Simulation, User's Guide, Département informatique et recherche opérationnelle, Université de Montréal, Jnavier 2003.

Sites utiles

1. Simulation et modélisations discrètes ;
http://ina.eivd.ch/ina/Collaborateurs/cez/Mod im/modsim.htmhttp
2. Douillet, ensait, http://193.43.37.48/~douillet/cours/oprea/node3.html
3. Nicolas Navet, -INRIA/TRIO, Evaluation des performances par simulation : introduction générale et présentation du logiciel QNAP2 , http://www.loria.fr/~nnavet
4. Cours de M. Hébuterne sur QNAP2 et la simulation à évènements discrets
http://www-rst.int-evry.fr/~hebutern/IT21/Simu.html
5. The MOSEL Language, http://www4.informatik.uni-erlangen.de/Projects/MOSEL/

6. http://cours.polymtl.ca/ifsos/GPSS/GPSS/)

7. N.Navet , Cours d'Evaluation des performances par simulation : introduction générale et présentation du logiciel QNAP2 http://www.loria.fr/~nnavet

8. Xiang Zeng et al, GloMoSim: a Library for parallel simulation of large-scale wireless networks, http://pcl.cs.ucla.edu/projects/glomosim.

9. Simulation software review. Discrete event simulation software package, http://www.tbm.tudelft.nl/webstaf/edwinv/SimulationSoftware/index.htm

10. Al-begain et al, The performance and reliability modelling language MOSEL and its application, International journal of Simulation vol.3, N°3-4, pp.66-79, 2004.