

Simulation à événements discrets

Thi-Mai-Trang Nguyen

Sorbonne Université – LIP6
trnguyen@lip6.fr

Plan

- Simulation à événements discrets
- Ordonnanceur
- Premier pas avec OMNeT++
- Exemple simple du réseau Tic-Toc

Simulation à événements discrets

- La simulation à événements discrets est la simulation d'un système dont l'état ne peut changer que lors d'instantanés temporels distincts où il y a un événement
- Un événement est une circonstance qui permet au système de changer d'état
- Exemple
 - Les files d'attente
- Deux approches pour la gestion du temps
 - Simulation avec avancement par incrément fixe
 - Simulation avec avancement par événement

Simulation avec avancement par incrément fixe (1)

- Une simulation par incrément fixe (i.e. time-driven simulation) exécute les événements après chaque intervalle de temps de valeur fixe
- La simulation avance d'un intervalle à un autre et exécute tous les événements (s'il y en a) jusqu'à ce qu'elle atteigne une certaine limite (e.g. le temps de fin de la simulation)



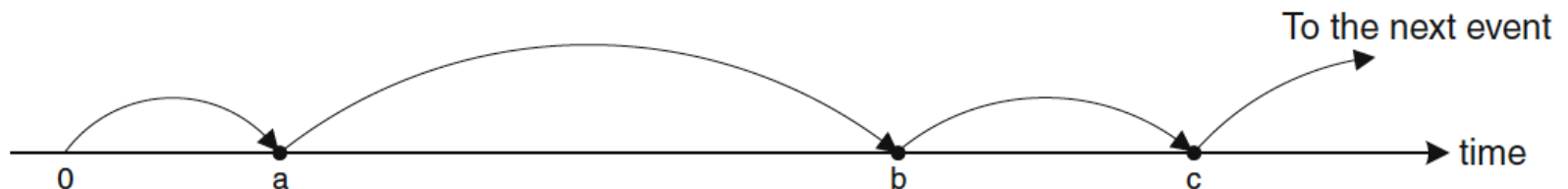
Simulation avec avancement par incrément fixe (2)

■ Pseudo-code

```
1 initialize {system states}
2 SimClock := startTime;
3 while {SimClock < stopTime}
4     collect statistics from current state;
5     execute all events that occurred during
6     [SimClock, SimClock + step];
7     SimClock := SimClock + step;
8 end while
```

Simulation avec avancement par événement (1)

- Une simulation avec avancement par événement (i.e. event-driven simulation) avance d'un événement à un autre en les exécutant jusqu'à ce que la simulation termine
- La simulation doit avancer toujours dans l'ordre chronologique
 - Chaque événement planifié dans la liste des événements doit avoir un temps de début qui est plus grand ou égal à celui de l'événement courant
 - Le prochain événement à exécuter par la simulation est toujours celui ayant le temps de début le plus petit



Simulation avec avancement par événement (2)

■ Pseudo-code

```
1 initialize {system states}
2 initialize {list of events}
3 while {state != finalState} % or while {this.event != Null}
4     expunge the previous event from list of events;
5     set SimClock := time of current event;
6     execute this.event
7 end while
```

Gestion des événements

- Dans une simulation à événements discrets, tous les événements de la simulation ne peuvent pas être créés lors du démarrage de la simulation
- Quand la simulation avance, un événement peut introduire un ou plusieurs nouveau(x) événement(s)
- Les nouveaux événements sont insérés dans une liste des événements qui sont rangés dans l'ordre chronologique
- Le processus continue jusqu'à ce que tous les événements soient exécutés ou le système arrive à un certain état (e.g. le temps de simulation atteint certaine valeur)
- C'est le travail de l'ordonnanceur des événements !

Ordonnanceur

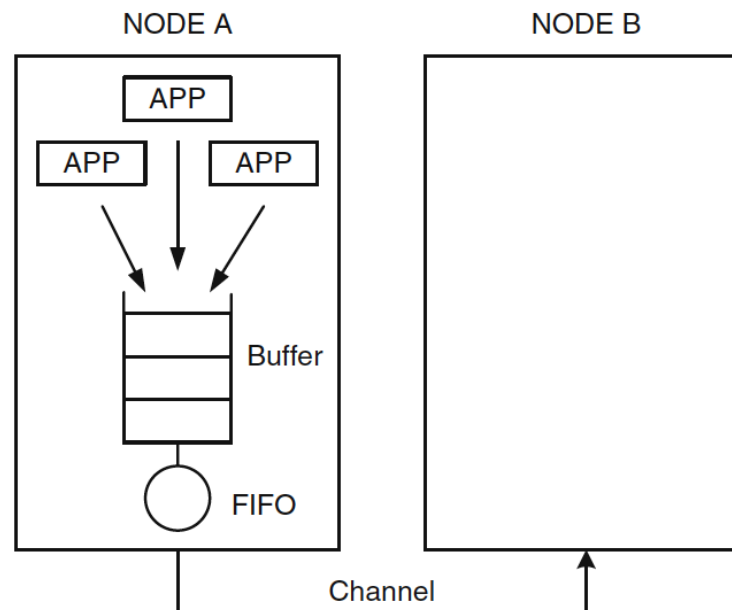
- L'ordonnanceur (i.e. échéancier) maintient une liste d'événements (encore appelé FES – Future Event Set ou FEL – Future Event List)
- Les opérations d'un ordonnanceur
 - Identification et invocation du prochain événement (tête de liste)
 - Insertion d'un événement futur dans la liste
 - Retrait d'un événement à la fin de son traitement
- Différents types d'ordonnanceur
 - Liste (List Scheduler)
 - Utiliser une structure de données sous la forme d'une liste ordonnée
 - Tas (Heap Scheduler) (OMNeT++)
 - Utiliser une structure de données sous la forme d'un arbre binaire dans lequel la valeur d'un nœud fils est toujours plus grande/petite ou égale à la valeur du nœud parent
 - Calendrier (Calendar scheduler) (par défaut dans NS-2)
 - Le temps est divisé en slots dont chacun contient une liste ordonnée des événements à exécuter dans ce créneau

Horloge de simulation

- L'horloge de simulation (i.e. simulation clock) indique le temps courant de la simulation (quand la simulation avance)
- La simulation suit la liste d'événements, exécute les événements l'un après l'autre et met à jours l'horloge de la simulation suivant les paramètres de l'événement exécuté

Exemple : la file d'attente à un serveur

- Une liaison filaire point-à-point unidirectionnel de A à B
- Les paquets viennent des applications du nœud A seront mis dans la file d'attente et seront transmis un par un sur la liaison



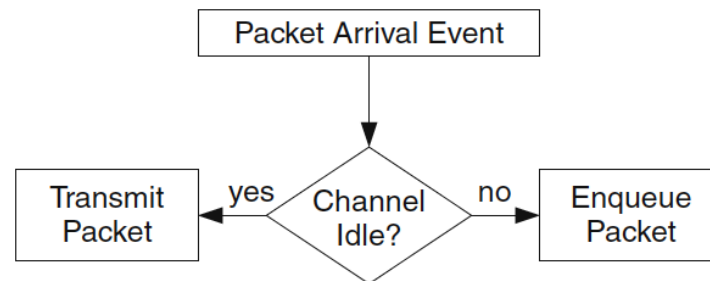
Concrétiser notre exemple (1)

- Entités
 - Serveur (i.e. l'entité qui prend un paquet de la file d'attente et l'envoie sur le canal)
 - « prêt » ou « occupé »
 - Paquet
 - Temps d'arrivée
 - Temps de service
 - Queue
 - « vide » ou « pas-vidé »
- Ressource
 - Le canal
- Etats du système
 - `num_system` : le nombre total de paquets dans le système
 - `channel_free` : l'état du serveur (prêt ou occupé)

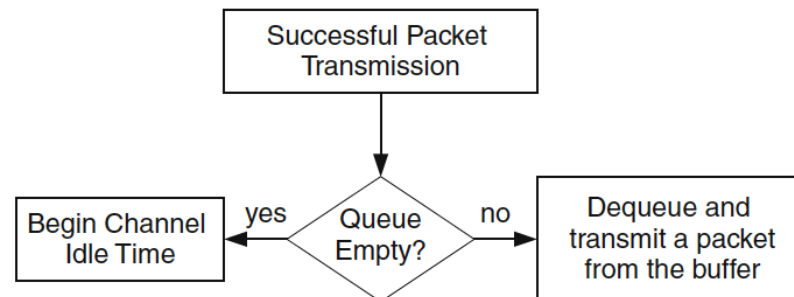
Concrétiser notre exemple (2)

■ Événements

- Pkt_arrival : l'arrivée d'un paquet



- Pkt_complete : la transmission d'un paquet est terminée



Concrétiser notre exemple (3)

- Métriques de performance (calculés à la fin de la simulation)
 - Temps d'attente (dans la file d'attente) moyen d'un paquet

$$\frac{\text{temps d'attente total de tous les paquets transmis}}{\text{nombre total des paquets transmis}}$$

- Délai moyen (temps d'attente + temps de service) d'un paquet

$$\frac{\text{temps total de passer dans le système de tous les paquets transmis}}{\text{nombre total des paquets transmis}}$$

- Taux d'utilisation moyen du serveur

$$\frac{\text{temps total d'occupation du serveur}}{\text{temps de simulation}}$$

Pseudo-code de l'exemple (1)

```
1  % Initialize system states
2  channel_free = true; %Channel is idle
3  num_queue = 0;      %Number of packets in queue
4  num_system = 0;      %Number of packets in system
5  SimClock = 0;        %Current time of simulation

6  %Generate packets and schedule their arrivals
7  event_list = create_list();

8  % Main loop
9  while {event_list != empty} & {SimClock < stopTime}
10     expunge the previous event from event list;
11     set SimClock := time of current event;
12     call current event;
13 end while
```

Pseudo-code de l'exemple (2)

```
14 %Define events
15 pkt_arrival(){
16     if(channel_free)
17         channel_free = false;
18         num_system = num_system + 1;
19         % Update "event_list": Put "successful packet tx event"
20         % into "event_list," T is random service time.
21         schedule event "pkt_complete" at SimClock + T;
22     else
23         num_queue = num_queue + 1; %Place packet in queue
24     num_system = num_queue + 1;
25 }
```


Pseudo-code de l'exemple (3)

```
26 pkt_complete(){
27     num_system = num_system - 1;
28     num_queue = num_queue - 1;
29     if(num_queue > 0)
30         schedule event "pkt_complete" at SimClock + T;
31     else
32         channel_free = true;
33         num_system = 0;
34         num_queue = 0;
35 }
```

Réalisation (1)

- Densité de probabilité des inter-arrivées et du temps de service

Time unit	Inter-arrival (probability mass)	Service (probability mass)
1	0.2	0.5
2	0.2	0.3
3	0.2	0.1
4	0.2	0.05
5	0.1	0.05
6	0.05	
7	0.05	

Réalisation (2)

■ Résultat de simulation de 10 paquets

Packet	Interarr. time	Service time	Arrival time	Service starts	Time spent in-queue	Packet trans- mission latency
1	-	5	0	0	0	5
2	2	4	2	5	3	7
3	4	1	6	9	3	4
4	1	1	7	10	3	4
5	6	3	13	13	0	3
6	7	1	20	20	0	1
7	2	1	22	22	0	1
8	1	4	23	23	0	4
9	3	3	26	27	1	4
10	5	2	31	31	<u>0</u>	<u>2</u>
					<u>10</u>	<u>3.5</u>

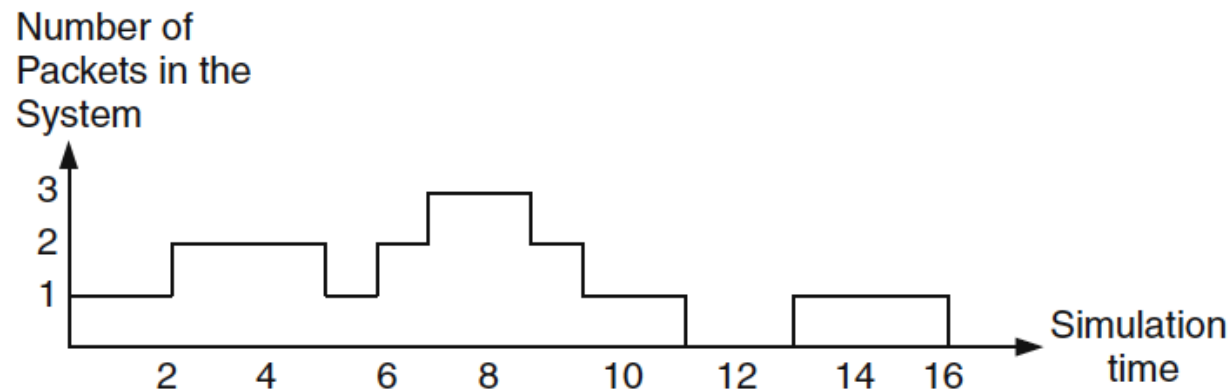
Réalisation (3)

- Les événements correspondant aux 5 premiers paquets

Event	Packet No.	Simulation clock
Arrival	1	0
Arrival	2	2
Completion	1	5
Arrival	3	6
Arrival	4	7
Completion	2	9
Completion	3	10
Completion	4	11
Arrival	5	13
Completion	5	16

Réalisation (4)

- Évolution du nombre de paquets dans le système



- Taux d'utilisation moyen du serveur

$$\frac{14}{16} = 0,875$$



OMNeT++

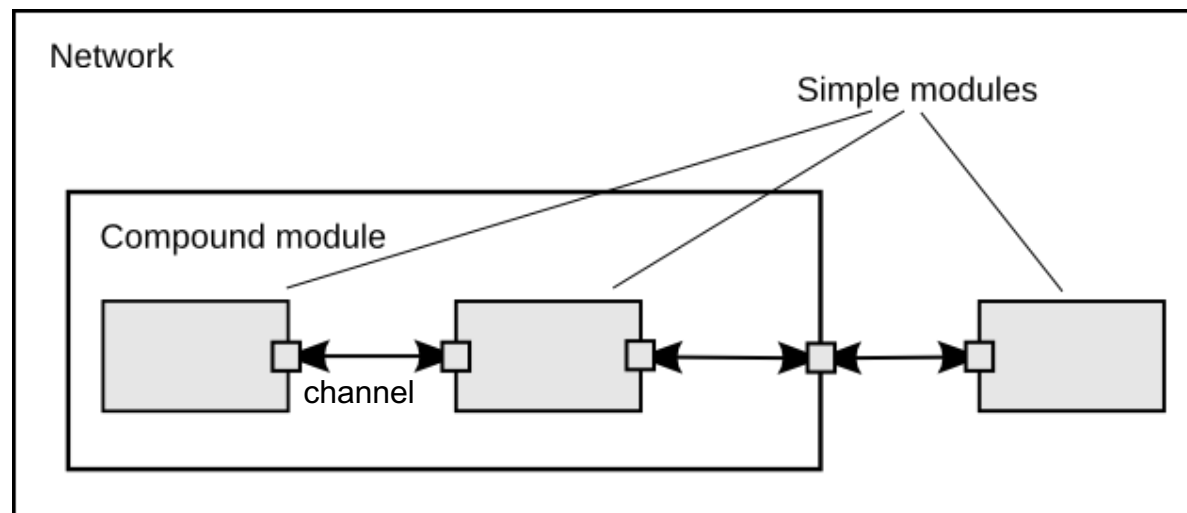
- Objective Modular Network Testbed in C++
- Pour les simulations à événements discrets
- Utiliser le langage C++
- Avec les modules spécialisés dans les réseaux (e.g. Internet)
- Très bien pour une plate-forme de simulation des réseaux dans les environnements académiques et industriels
- <http://www.omnetpp.org>

OMNeT++ lors du lancement



Architecture de OMNeT++ (1)

- 4 entités de base dans une simulation
 - Module simple (simple module)
 - Canal (channel)
 - Module composé (compound module)
 - Réseau (network)



Architecture de OMNeT++ (2)

- Module simple
 - Brique de base
- Canal
 - Interconnecter les modules
- Les modules se communiquent (échanger des messages) par un canal et y accéder à travers des ports (i.e. « gates »)

Deux langages

- C++
 - Implémenter les fonctionnalités et composants spécifiques
- NED
 - Décrire les modules
 - Topologie du réseau
 - Configuration des paramètres d'une simulation

Langage NED

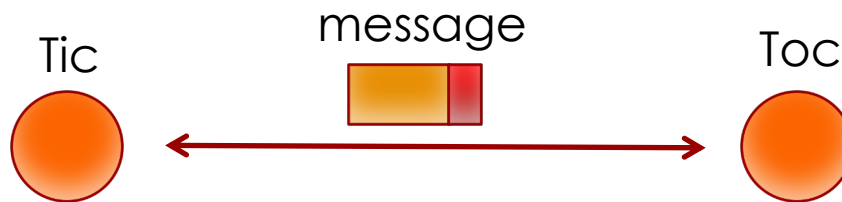
- Un langage spécifique à OMNeT++ pour décrire la structure d'un réseau (NED = Network Description)
- Exemple



```
//  
// A network  
//  
network Network  
{  
    submodules:  
        node1: Node;  
        node2: Node;  
    connections:  
        node1.port++ <--> {datarate=100Mbps;} <--> node2.port++;  
}
```

Exemple: Réseau Tic-toc

- Tic-Toc est un réseau ayant deux nœuds (Tic et Toc)
- Tic envoie un message à Toc qui le renvoie et ainsi de suite



- Deux manières de créer les simulations
 - Par lignes de commande et un éditeur de texte au choix
 - Par l'IDE de OMNeT++ (à découvrir dans le TP)

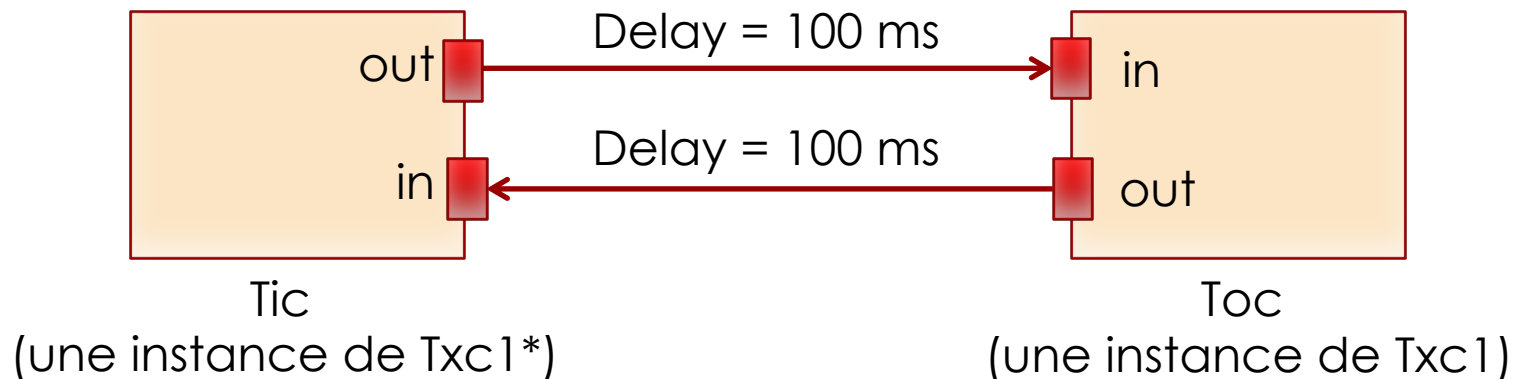
Réaliser le réseau Tic-Toc en utilisant des lignes de commande

- Créer un dossier appelé `tictoc`

```
$ mkdir tictoc
```

```
$ cd tictoc
```

- Editer un fichier `tictoc1.ned` pour décrire la topologie du réseau `Tictoc1`



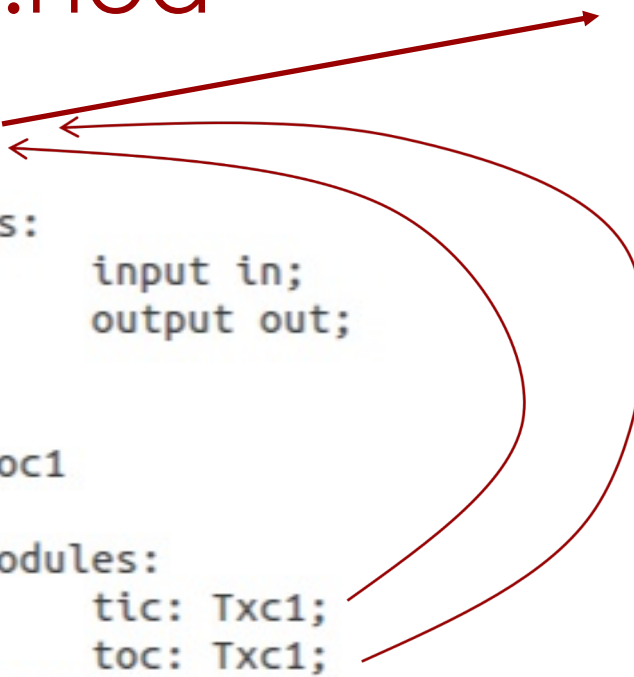
** `Txc1` est un module simple ayant deux ports (`in` et `out`) à implémenter*

tictoc1.ned

(classe Txc1 dans le fichier txc1.cc)

```
simple Txc1
{
    gates:
        input in;
        output out;
}

network Tictoc1
{
    submodules:
        tic: Txc1;
        toc: Txc1;
    connections:
        tic.out --> { delay = 100ms; } --> toc.in;
        tic.in <-- { delay = 100ms; } <-- toc.out;
}
```



Comprendre le fichier `tictoc1.ned`

- Description du réseau `Tictoc1`
 - `Tictoc1` est un réseau formé de 2 sous-modules : `tic` et `toc`
 - `tic` et `toc` sont des instances d'un module simple qui s'appelle `Txc1`
 - `Txc1` est un module simple ayant un port d'entrée appelé « `in` » et un port de sortie appelé « `out` »
 - Le port de sortie de `tic` est connecté au port d'entrée de `toc` par un canal ayant un délai de 100 ms
 - Le port de sortie de `toc` est connecté au port d'entrée de `tic` par un canal ayant un délai de 100 ms
- Ensuite, il faut créer un fichier `txc1.cc` pour implémenter les fonctionnalités du module simple `Txc1`

txc1.cc

```
#include <string.h>
#include <omnetpp.h>
```

```
class Txc1 : public cSimpleModule
{
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};
```

```
Define_Module(Txc1);
```

→ (pour l'intégrer/enregistrer avec les autres Modules de OmneT++)

```
void Txc1::initialize()
{
    if (strcmp("tic", getName()) == 0)
    {
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    send(msg, "out");
}
```


Comprendre le fichier txc1.cc

- Tout module simple doit être implémenté comme une sous-classe de la classe `cSimpleModule` et doivent être enregistrés auprès de OMNeT++ via le macro `Define_Module()`
- Nous allons redéfinir deux méthodes définies par la classe `cSimpleModule`
 - `initialize()`
 - `handleMessage()`
- La méthode `initialize()` sera invoquée une seule fois lors de l'instanciation d'un objet
- La méthode `handleMessage()` sera invoquée à chaque fois un message arrive au module

Fichier de configuration

- Pour paramétrer la simulation, il faut créer un fichier `.ini` (souvent nommé `omnetpp.ini`)
- Fichier `omnetpp.ini`

```
[General]
```

```
network = Tictoc1
```

- Dossier `.../tictoc` à présent
 - `omnetpp.ini`
 - `tictoc1.ned`
 - `txc1.cc`

Compiler et lancer la simulation

- Créer le Makefile

```
$ opp_makemake
```

- Compiler et créer le fichier exécutable

```
$ make
```

- Exécuter le programme

```
$ ./tictoc
```

Interface de simulation

OMNeT++/Tkenv - General #0 - omnetpp.ini - /home/trang/omnetpp/tictoc

File Simulate Inspect View Help

General #0: Tictoc1 Event #1 t=0.1s Msg stats: 1 scheduled / 2 existing / 2 created

Next: tictocMsg (cMessage, id=0) In: Tictoc1.toc (Txc1, id=3) At: last event + 0.1s

Tictoc1 (Tictoc1) (id=1)

scheduled-events (cMessageHeap)

Tictoc1

tic

toc

Zoom: 1.00x

(Tictoc1) Tictoc1

Fields Contents (2)

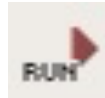
Class	Name	Info
Txc1	tic	id=2
Txc1	toc	id=3

Event# Time Src/Dest Name

tic --> toc

A vous de jouer !

- Cliquer sur le bouton « Run » pour lancer la simulation



- Cliquer sur le bouton « Stop » pour arrêter la simulation



- Cliquer sur le bouton « Conclude simulation » pour invoquer la méthode finish() dans tous les modules



- Nous voyons les nœuds Tic et Toc échanger le message « tictocMsg »

Résultat

OMNeT++/Tkenv - General #0 - omnetpp.ini - /home/trang/omnetpp/tictoc

File Simulate Inspect View Help

General #0: Tictoc1 Event #14 t=1.4s Msg stats: 1 scheduled / 15 existing / 15 created

Next: n/a In: n/a At: n/a

Tictoc1 (Tictoc1) (id=1)

scheduled-events (cMessageHeap)

Tictoc1

(cMessage) tictocMsg

toc

Zoom: 1.00x

(cMessage) tictocMsg

Fields Contents (0)

tictocMsg (cMessage)

- controlInfo = NULL (cObject)
- base
- message
- sending

Event#	Time	Src/Dest	Name
-	0	tictoc --> toc	
#1	0.1	toc --> tictoc	
#2	0.2	tictoc --> toc	
#3	0.3	toc --> tictoc	tictocMsg
#4	0.4	tictoc --> toc	tictocMsg
#5	0.5	toc --> tictoc	tictocMsg
#6	0.6	tictoc --> toc	tictocMsg
#7	0.7	toc --> tictoc	tictocMsg
#8	0.8	tictoc --> toc	tictocMsg
#9	0.9	toc --> tictoc	tictocMsg
#10	1	tictoc --> toc	tictocMsg
#11	1.1	toc --> tictoc	tictocMsg
#12	1.2	tictoc --> toc	tictocMsg
#13	1.3	toc --> tictoc	tictocMsg