Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

**MASTER EN SCIENCES INFORMATIQUES**

# Implémentation d'un simulateur d'un système de distribution de contenu vidéo sur internet

Choquet, Olivier

*Award date:*
2011

Link to publication

**General rights**
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

**Take down policy**
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026

# Implémentation d'un simulateur d'un système de distribution de contenu vidéo sur internet

Choquet Olivier

juin 2011

# Table des matières


Remerciements . . . . . 1

**1 Distribution vidéo : Inventaire des systèmes** **2**
1.1 Description des services vidéos . . . . . 2
1.2 Liste de catégories des services vidéos . . . . . 3
1.3 Différenciation des services vidéos . . . . . 3
1.3.1 Network Type . . . . . 5
1.3.2 Quality of Service . . . . . 5
1.3.3 Multipoint Method . . . . . 6
1.3.4 Key Protocols . . . . . 7
1.3.5 Viewing Devices . . . . . 11
1.3.6 Program Choices . . . . . 12
1.3.7 User Experience . . . . . 12
1.3.8 Channel Change Time . . . . . 13
1.3.9 Rewind / Fast-Forward . . . . . 13
1.3.10 Production Values . . . . . 14
1.3.11 Content Types . . . . . 14
1.3.12 Program Library . . . . . 15
1.3.13 Ownerships Rights . . . . . 15
1.3.14 Revenues Models . . . . . 16
1.3.15 Examples Providers . . . . . 16
1.4 Description du système de distribution de contenu vidéo étudié . . . . . 17
1.5 Catégorie du service vidéo implémenté par le simulateur . . . . . 18
1.6 Conclusion . . . . . 20

**2 Introduction à la simulation par événements discrets** **21**
2.1 Classification des modèles de simulation à événements discrets . . . . . 21
2.2 Terminologie de la simulation par événements discrets . . . . . 23
2.3 Démarche générale d'une simulation par événements discrets : présentation des éléments essentiels . . . . . 25
2.3.1 La génération des données d'entrée . . . . . 25
2.3.2 La simulation . . . . . 28
2.3.3 Analyse des résultats . . . . . 29

**3 Analyse et Modélisation du système de distribution de contenu vidéo** **32**
3.1 Méthodologie de la simulation par événements discrets . . . . . 32
3.2 Analyse du système de distribution de contenu vidéo . . . . . 34
3.2.1 Objectifs de l'analyse . . . . . 34

3.2.2 Modèle conceptuel . . . . . 34
3.2.3 Modèle de spécifications . . . . . 35

**4 Analyse pour la construction du simulateur : Modèle informatique 37**
4.1 Objectifs du simulateur . . . . . 37
4.2 Programmation par événements discrets . . . . . 38
4.2.1 Construction d'un modèle de simulation *Next-Event* . . . . . 39
4.2.2 Algorithme de simulation *Next-Event* . . . . . 39
4.3 Analyse pour la construction du simulateur . . . . . 39
4.3.1 Variables d'états . . . . . 39
4.3.2 Découpe en événements . . . . . 41
4.3.3 Algorithmes pour le changement d'état . . . . . 41

**5 Implémentation du simulateur 47**
5.1 SSJ . . . . . 47
5.1.1 Description générale de SSJ . . . . . 47
5.1.2 Notation UML : Diagramme de classes . . . . . 48
5.1.3 Description détaillée de SSJ . . . . . 50
5.2 Diagramme de classes du simulateur . . . . . 54
5.3 Interface graphique . . . . . 58
5.4 Mécanismes de performances et d'efficacité appliqués au simulateur . . . . . 61

**6 Vérification et Validation du simulateur 63**
6.1 Vérifier le simulateur . . . . . 63
6.2 Valider le simulateur . . . . . 64

**7 Modification du modèle conceptuel 67**
7.1 Modifications apportées . . . . . 67
7.2 Étude numérique . . . . . 70
7.2.1 Présentation des scénarios . . . . . 70
7.2.2 Résultats des simulations . . . . . 71
7.3 Conclusion . . . . . 79

**A Guide de l'utilisateur 80**
A.1 Installation du simulateur . . . . . 80
A.2 Réaliser une simulation . . . . . 80
A.3 Voir les résultats d'une simulation . . . . . 82
A.4 Enregistrer les résultats d'une simulation . . . . . 84
A.5 Comparer les résultats de plusieurs simulations . . . . . 85

**B Sauvegarde XML 86**

**C Données test 91**

**Bibliographie 93**

# Table des figures


1.1 Liste de catégories vidéos issue de [7, Chapitre 2, page 18] . . . . . 4
1.2 Avantages et Inconvénients des différentes techniques de diffusion vidéo. Figure issue du livre : [7, Chapitre 11, page 154] . . . . . 10

2.1 Types de modèles de systèmes. Figure issue de [3, Chapitre1, page 2] . . . . . 22
2.2 Modèle conceptuel d'une file d'attente simple. Figure issue de [3, Chapitre 1, page 11] . . . . . 23
2.3 Génération d'une variable aléatoire discrète. Figure issue de [3, Chapitre 6, page 237] . . . . . 28
2.4 Illustration de l'état stationnaire issue du simulateur implémenté : comparaison entre une simulation de 1000 secondes et une simulation de 10000 secondes. . . 29
2.5 Exemple de courbe empirique avec la fonction de distribution cumulative(FDC) théorique correspondante. Figure issue de [3, Chapitre 4, page 170] . . . . . 31

3.1 Modèle conceptuel du système de distribution vidéo . . . . . 34
3.2 Résultats de l'étude : comparaison entre la loi exponentielle, erlang et l'échantillon de l'étude. Figure issue de [5] . . . . . 36

4.1 Diagrammes d'activités : légende . . . . . 42
4.2 Comportement du simulateur : schéma général . . . . . 42
4.3 Comportement du simulateur : événement *Arrival* . . . . . 43
4.4 Comportement du simulateur : événement départ normal et avec impatience theta1 . . . . . 45
4.5 Comportement du simulateur : événement *DepartureImpatience2* . . . . . 46

5.1 Diagramme de classe de la librairie SSJ (partie Générateur de nombres aléatoires) 51
5.2 Diagramme de classe de la librairie SSJ (partie Événement) . . . . . 52
5.3 Diagramme de classe de la librairie SSJ (partie Collecte de statistiques) . . . . 53
5.4 Diagramme de classe du simulateur . . . . . 55
5.5 Interface graphique : réaliser une nouvelle simulation . . . . . 59
5.6 Interface graphique : comparer des simulations . . . . . 60

6.1 Analyse du fichier log par l'outil *Chainsaw* pour le client 1 . . . . . 64
6.2 Comparaison entre le graphique Fig2a de l'étude [5] et le graphique issu du simulateur . . . . . 65
6.3 Comparaison entre le graphique Fig4a de l'étude [5] et le graphique issu du simulateur . . . . . 66

6.4 Comparaison entre le graphique Fig4b de l'étude [5] et le graphique issu du simulateur . . . . . 66

7.1 Scénario1 : répartition des temps de séjours . . . . . 73
7.2 Scénario2 : répartition des temps de séjours . . . . . 74
7.3 Scénario2 : nombre de clients dans le système . . . . . 75
7.4 Scénario 3 : répartition des temps de séjours . . . . . 77
7.5 Scénario 3 : nombre de clients dans le système . . . . . 78

**Résumé**

L'objectif de ce mémoire est l'implémentation d'un simulateur d'un système de distribution de contenu vidéo sur internet. Le simulateur fournit un outil dynamique capable d'analyser un système de distribution de contenu vidéo par le biais de simulations.

Ce système a été analysé dans l'article [5] suivant différentes variables de performances notamment l'impatience des utilisateurs. Cette étude a été réalisée selon une approche mathématique et suivant une hypothèse sur la distribution de la taille des fichiers vidéos. Bien que la distribution de la taille des fichiers semblait s'adapter au mieux avec une distribution de type Erlang, l'étude a fait l'hypothèse d'utiliser une loi Exponentielle pour représenter la taille de fichiers. Le but du simulateur est d'implémenter le modèle de simulation décrit par l'étude afin de vérifier dans un premier temps les résultats de cette étude. Dans un second temps, l'objectif est de modifier l'hypothèse sur la distribution de la taille des fichiers afin de tirer des conclusions à propos de l'hypothèse choisie par l'étude.

Pour ce faire, ce document est structuré de la manière suivante. Un inventaire et une classification des systèmes de distribution de contenu vidéo est réalisé afin de situer le système de contenu vidéo qui nous intéresse parmi le systèmes existants. Ensuite, nous nous concentrons sur ce système afin de réaliser l'implémentation du simulateur. Le simulateur est basé sur la simulation par événements discrets. C'et pourquoi celle-ci sera tout d'abord présentée et expliquée. Il s'en suit l'analyse du modèle conceptuel du système de distribution vidéo. Nous poursuivons par expliquer l'analyse du modèle informatique effectué. Finalement, nous réalisons différentes simulations avec la distribution Exponentielle et Erlang afin de les comparer.

# Remerciements

Je tiens à remercier dans un premier temps, toute l'équipe pédagogique et administrative des Facultés Universitaires Notre Dame de la Paix.

Je remercie également Madame Remiche pour l'aide et les conseils qu'elle m'a apportés durant l'élaboration de ce mémoire.

# Chapitre 1

# Distribution vidéo : Inventaire des systèmes

## Sommaire


| | | |
|---|---|---|
| **1.1** | **Description des services vidéos** | **2** |
| **1.2** | **Liste de catégories des services vidéos** | **3** |
| **1.3** | **Différenciation des services vidéos** | **3** |
| **1.4** | **Description du système de distribution de contenu vidéo étudié** | **17** |
| **1.5** | **Catégorie du service vidéo implémenté par le simulateur** | **18** |
| **1.6** | **Conclusion** | **20** |


L'objectif de ce chapitre est de faire l'inventaire des systèmes de distributions vidéos et de les classifier selon une liste de critères. Nous appellerons services vidéos dans la suite de ce document ces systèmes de distribution vidéos.

Pour classifier ces services vidéos, nous reprenons une liste de catégories de services vidéos existante. Cette liste a été développée par Wes Simpson et Howard GreenField [7], deux spécialistes des télécommunications et des services vidéos.

Ensuite, nous expliquons les différents critères de classification de cette liste. Nous concluons en énonçant à quelle catégorie appartient le service vidéo à la base de l'étude [5] et pour laquelle nous développons un simulateur.

## 1.1 Description des services vidéos

La *Video On Demand*(VOD) et l'*IP Television*(IPTV) sont actuellement en plein essor et on ne compte plus le nombre d'offres différentes proposées par les entreprises commerciales. On peut citer par exemple : Belgacom TV , la vidéo à la demande de Voo, OnlineTVRecorder [1], YouTube [2], Zattoo [3], ...

Actuellement vu la diversité des services vidéos proposés, aussi bien du point de vue technique que culturelle, il est difficile de savoir ce qu'une personne entend par IPTV ou VOD ou encore service vidéo sans avoir de précisions. La personne parle-t-elle d'un service vidéo tel que YouTube avec un contenu généré par les visiteurs et une architecture basée sur HTTP, ou parle-t-elle d'un service vidéo tel que le service VOD de Belgacom TV permettant le contrôle du flux vidéo ?

Pour lever la confusion, les services vidéos peuvent être décrits selon une liste de critères et classer en quatre catégories de base.

1. `http://www.onlinetvrecorder.com`
2. `http://www.youtube.com`
3. `http://zattoo.com`

## 1.2 Liste de catégories des services vidéos

Wes Simpson et Howard Greenfield, deux spécialistes des télécommunications et services vidéos proposent quatre catégories de base de services vidéos, à savoir

- *IPTV*
- *IPVOD*
- *Internet TV*
- *Internet Video*

Cette liste est issue de [7, Chapitre 2, page 18].
Voici une brève description de ces différentes catégories.

**IPTV** Cette catégorie regroupe les services vidéos qui proposent un ensemble de chaînes de télévision via les technologies IP. Il s'agit de l'adaptation de la télévision analogique au numérique. Les acteurs de cette catégorie sont généralement les fournisseurs d'accès qui adaptent leurs réseaux pour le support du protocole IP. Ils distribuent alors un ensemble de chaînes de télévision de la même manière qu'ils le faisaient pour la télévision câblée traditionnelle, mais à l'aide du protocole IP cette fois.

**IPVOD** Cette catégorie regroupe les services fournissant de la vidéo à la demande aux clients c'est-à-dire la possibilité pour le client de visionner une vidéo instantanément. Dès que le client demande une vidéo, celle-ci est visionnée directement sans temps d'attente ou temps de téléchargement. Les fournisseurs d'accès offrent également cette catégorie de service. D'autres acteurs présents sur Internet proposent également ce service. Ils mettent à disposition de leurs clients un catalogue de vidéos à visionner moyennant rémunération.

**Internet TV** Cette catégorie regroupe des services vidéos dits temps réel, présents sur le Web. Ils mettent à la disposition des internautes un ensemble de chaînes de télévision qui peuvent être vues en temps réel.

**Internet Video** Cette catégorie utilise Internet pour la distribution vidéo. Ces services vidéos sont accessibles via un site web et proposent un catalogue de vidéos à visionner. Ces vidéos sont souvent gratuites ou produites par les internautes.

Il existe une catégorie de services vidéos que l'on appelle *Real-Time Interactive Audio and Video*. Cette catégorie regroupe des services vidéos tels que la téléphonie sur IP ou encore la vidéoconférence. La liste des catégories de services vidéos issue de [7] que nous suivons ici n'en parle pas.

Bien entendu ces catégories ne sont qu'une aide pour se repérer parmi les services vidéos actuels les plus répandus. Les frontières entre les différentes catégories sont parfois minces. Des services vidéos peuvent se retrouver à cheval sur plusieurs catégories.

## 1.3 Différenciation des services vidéos

Regardons maintenant ces catégories au travers de différents critères. Une explication des critères suit le tableau.

| **Services Attributes** | **IPTV** | **IPVOD** | **Internet TV** | **Internet Video** |
|---|---|---|---|---|
| Network type | Private Network | Public Network | Public Network | Public Network |
| Quality of service | Managed Qos | Unmanaged Qos | Unmamaged Qos | Unmanaged Qos |
| Multipoint method | True multicasting | Unicasting | Replicated unicasting | Unicasting |
| Key protocols | True Streaming RTP over UDP | Progressive download and play | HTTP streaming, progressive download and play | HTTP Streaming, progressive download and play |
| Viewing devices | STB with television | STB with television or PC | PC, Mobile or Network appliance | PC, Mobile or Network appliance |
| Program choices | Hundreds of channels of continuous TV | Thousand of discrete video files | Thousand of channels of continuous TV | Millions of discrete video files |
| User experience | Similar to broadcast or cable tv | Similar to DVR or VOD | Similar to web surfing | Similar to web surfing |
| Channel change time | Quick : 1-2 sec. | Reasonable : 5-10 sec. | Slow : 10-20 sec. | Slow : 10-20 sec. |
| Rewind/Fast-forward | No | Yes | No | Yes |
| Production values | Professionally produced | Professionally produced | Professionally produced | User generated |
| Content types | Live or Prerecorded | Prerecorded only | Live or Prerecorded | Prerecorded only |
| Program library | Walled content garden | Walled content garden | Worldwide reach, quality varies | Viewer beware |
| Ownership rights | Strong, with digital right management | Strong, often with DRM | Fairly strong | Weak or inexistent, frequent copyright violations |
| Revenues models | Paid by subscription | Subscription, fee per episode or ads | Often free or with advertising | Often free or with advertising |
| Examples providers | Local Telcos, AT-T | Netflix, Hulu, ABC.com | NASA.tv, Local TV broadcasters | Youtube, Facebook |

FIGURE 1.1 – Liste de catégories vidéos issue de [7, Chapitre 2, page 18]

Chaque critère est maintenant expliqué. Ces critères explorent plusieurs facettes des services vidéos. Ils sont soit techniques, culturels ou économiques. Cette liste de critères n'est pas exhaustive et nous pouvons bien sûr rechercher d'autres critères pour différencier les services vidéos.

### 1.3.1 Network Type

Le type de réseau employé pour les services vidéos peut être privé ou public. Un réseau privé est la propriété d'un individu, d'une organisation ou d'une entreprise. Les réseaux privés utilisent un espace d'adressage privé conformément à la *Request For Comment* (RFC) 1918. Le propriétaire d'un réseau privé définit lui-même les règles de son réseau. Il possède le contrôle sur tous les appareils connectés à son réseau. Il attribue les adresses IPs lui-même et décide de l'autorisation ou de l'interdiction des protocoles réseaux ou services dans son réseau. Dans le cas des services vidéos dans un réseau privé l'emploi du protocole réseau RTP (voir 1.3.4) est alors autorisé.

Le réseau public par excellence est aujourd'hui Internet. Ce réseau n'est la propriété d'aucune organisation. Cependant des règles ont été définies sur l'attribution des adresses IPs et les noms de domaines. Chaque continent dispose d'un organisme en charge de l'attribution des adresses IPs. Toute personne désirant un accès à Internet doit alors se tourner vers ces organismes.

Internet est constitué de routeurs/pare-feux appartenant à diverses entreprises. Chacune de ces entreprises, généralement des fournisseurs d'accès Internet, fixent leurs propres règles. Les services vidéos utilisant Internet doivent donc s'accommoder aux différentes règles en vigueur. Ils ne peuvent pas toujours, par exemple, utiliser le *multicast* (voir section 1.3.3) et le protocole RTP (voir section 1.3.4). Les services vidéos se tournent alors vers le protocole HTTP et la diffusion *unicast* (voir section 1.3.3).

### 1.3.2 Quality of Service

La qualité de service permet de s'assurer que chaque spectateur recevra un flux vidéo de manière continue et de bonne qualité. Dans les réseaux informatiques, la qualité de service est réalisée par un mécanisme de priorité entre les services. Chaque service se voit attribuer une priorité ce qui permet aux routeurs de donner la priorité à certains types de paquets. Ceci est relativement simple à implémenter dans un réseau privé où nous disposons du contrôle de tous appareils connectés au réseau. Nous pouvons alors définir une politique de qualité de service et l'appliquer à tous les appareils du réseau.

Sur Internet, il n'existe aucun mécanisme pour forcer une politique de qualité de service à travers tout le réseau. Ceci s'explique par le fait que si cette possibilité existait, tout le monde considérerait son traffic comme prioritaire et donc le mécanisme de priorité deviendrait inutile.

C'est pourquoi en ce qui concerne les catégories de services vidéos, nous retrouvons de la qualité de service uniquement pour la catégorie IPTV où le réseau appartient entièrement au fournisseur d'accès.

### 1.3.3 Multipoint Method

Pour distribuer du contenu vidéo, il existe trois méthodes : le *multicasting*, l'*unicasting* et le *replicated unicasting*.

L'*unicasting* est une méthode de connexion réseau point à point. Chaque paquet de données est transmis depuis une source unique vers une destination unique.
La technique *unicast* est utilisée pour la distribution vidéo sur Internet. Le serveur vidéo dialogue séparément avec chaque client. Ceci permet une grande souplesse aux services vidéos. Le serveur peut répondre et s'adapter aux interactions de chaque client de manière individuelle. Les clients peuvent donc interagir directement avec le service vidéo via notamment les fonctionnalités de contrôle de la lecture vidéo telles que l'avance rapide, la pause, le retour en arrière. Cependant cette technique nécessite de maintenir entre chaque client et le serveur un dialogue. Pour chaque client, le serveur doit maintenir un dialogue. Le nombre de clients influence donc fortement la charge de travail du serveur.

Une autre technique couramment employée par les services vidéos est le *multicasting*. Le *multicasting* est une méthode de connexion réseau multi-points. Chaque paquet de données est transmis depuis une source unique vers plusieurs destinations. Le *multicasting* peut s'effectuer à différents niveaux du modèle OSI (Pour plus de détails sur le modèle OSI, consultez [2, Chapitre 1, page 50]). Le *multicast* est implémenté au niveau de la couche réseau du modèle OSI via une classe d'adresses IPs particulière destinée à l'emploi du *multicast*. Cependant il faut un protocole pour gérer les clients qui s'ajoute au groupe *multicast*, contrôler le début et la fin d'une distribution *multicast*. Ceci est réalisé par le protocole *Internet Group Management Protocol* (IGMP) (Pour plus de détails sur le protocole IGMP, consultez [2, Chapitre 4, page 410]. Le *multicast* se base sur le principe suivant : à des points de routage stratégique des copies des paquets de données à l'intérieur même du réseau sont générées à destination de chaque membre du groupe multicast.
Notons également que la distribution vidéo peut aussi s'opérer via les *multicast overlay networks*. Il s'agit de serveurs déployés sur Internet et qui permettent de réaliser la distribution vidéo en *multicast*. Ces réseaux sont typiquement construits au-dessus des couches TCP/IP. Ils sont constitués de nœuds reliés entre eux par un lien logique. Contrairement au *multicast* IP présenté ci-dessus, ici le *multicast* est réalisé par la couche application du modèle OSI et non pas dans la couche réseau.
La technique du *multicasting* se prête bien à la diffusion en masse. Le serveur vidéo ne doit envoyer qu'un seul paquet, le réseau se chargeant lui-même de réaliser des copies à destination de chaque client. La charge de travail du serveur vidéo n'augmente donc pas en fonction du nombre de clients. C'est pourquoi cette technique est souvent employée pour la diffusion de l'*IPTV*. Cependant, l'usage du *multicast* sur Internet n'est pas pris en charge actuellement par crainte de surcharger le réseau. Cette technique ne peut donc pas s'utiliser sur Internet.

Enfin il existe une autre technique : le *replicated unicasting*. Cette méthode tente de simuler une distribution *multicast* sur Internet. Le *multicast* n'étant pas supporté sur Internet, le serveur crée lui-même des copies des paquets de données qu'il envoie ensuite aux différents clients. Cette technique est utilisée par les systèmes *Internet TV*.

Pour plus de détails sur les techniques de diffusion et de transports, nous nous référons à [7, Chapitre 5, page 76].

### 1.3.4 Key Protocols

Les services vidéos utilisent tous le protocole *Internet Protocol* (IP). L'emploi de ce protocole présente de nombreux avantages. Le réseau IP offre une excellente couverture géographique et technologique. Le protocole IP est la première technologie en terme de couverture mondiale, il permet d'assurer le lien entre de nombreuses technologies (WIFI, DSL, fibre optique, ...) de niveau inférieur et le niveau applicatif. Le réseau IP offre aussi beaucoup de souplesse. L'accès à ce protocole est public et il est bien documenté. De nombreuses applications se développent donc sur IP et en particulier les services vidéos. Le coût du matériel ne cesse aussi de diminuer dû à sa grande exploitation ce qui n'est pas négligeable économiquement pour les entreprises.

Quelques inconvénients au mariage des applications vidéos et du réseau IP sont cependant à signaler. Il faut savoir que tout message est transporté sous forme de paquets de données. Or le réseau IP est *best-effort*, cela signifie qu'il n'existe pas de priorité entre les différents flux de paquets envoyés. Un flux de paquets d'une application multimédia aura la même priorité que le flux de paquets d'une application de recherche sur Internet par exemple. Les applications multimédia étant très sensibles aux délais point à point contrairement aux applications classiques, il en résulte un véritable défi pour les services vidéos de livrer à temps les paquets de données.
Un autre problème que l'on rencontre dans les réseaux IPs est le fait que les paquets de données envoyés à une cadence déterminée n'arrive pas nécessairement à cette même cadence. Ce phénomène est appelé *packet jitter*. Il s'explique par le fait que les réseaux IPs sont constitués de routeurs disposant de systèmes de files d'attentes, les délais à l'intérieur de ces files d'attentes étant variables. Ces différents problèmes peuvent perturber fortement la fluidité et la qualité d'une vidéo visionnée en temps réel.

Les services vidéos utilisent fréquemment deux techniques pour diffuser du contenu vidéo : la diffusion par téléchargement et la diffusion par *streaming*. La technique de diffusion par téléchargement impose le téléchargement complet du fichier avant sa lecture.
La technique du *streaming* transfère les données sous la forme d'un flux régulier et continu. Le *streaming* permet donc de visualiser un contenu vidéo en temps réel.

On peut définir quatre types de distribution vidéo couramment employés :
- le *True Streaming*
- le *Download And Play*
- le *Progressive Download And Play*
- le *Podcasting*

### True Streaming

Le *True Streaming* consiste à lire un fichier vidéo à la volée sans temps d'attente. Un fichier vidéo de 10 minutes sera livré en 10 minutes ni plus, ni moins. Cette technique nécessite l'emploi d'un serveur de streaming et utilise le protocole *Real-time Transport Protocol* (RTP) ainsi que son compagnon le protocole *Real-time Transport Control Protocol* (RTCP). Ces protocoles sont clairement ciblés pour le multimédia et offrent des possibilités de contrôle de la vidéo en cours de diffusion. Le protocole RTP a pour but de transporter les paquets vidéos. Il utilise comme protocole de transport UDP ce qui permet l'emploi du *multicast* (voir 1.3.3). Les paquets vidéos transportés par RTP contiennent entre autres des informations sur l'encodage du fichier vidéo, un numéro de séquence afin de détecter les pertes de paquets, une date et heure d'envoi pour éviter les problèmes de *Packet Jitter* et un identifiant de la source vidéo. Le protocole RTCP, quant à lui, collecte des statistiques de réception des paquets chez les clients afin de permettre à la source vidéo de s'adapter. Pour plus de détails sur ces protocoles, référez-vous à [2, Chapitre 7, page 623].

Le *True Streaming* s'emploie typiquement pour la diffusion d'émissions de radio ou de télévision en temps réel.

### Download and Play

Cette technique consiste à télécharger entièrement le fichier vidéo avant de le lire. Dans ce cas il faut prendre en compte le temps de téléchargement qui peut varier suivant le débit de la connexion réseau. Sur un réseau rapide de 100 Mbits/s, un fichier vidéo de 100 MegaBytes sera téléchargé en 8,39 secondes tandis que sur un réseau plus lent de 2 Mbits/s le fichier sera téléchargé en 7 minutes. A ce temps de téléchargement il faut ajouter le temps de lecture du fichier vidéo. Un fichier vidéo de 10 minutes sera donc visionné entièrement en 10 minutes auquel il faut ajouter un temps de téléchargement variable.

Cette technique est couramment employée pour fournir de la vidéo à la demande. Les puristes diront qu'il ne s'agit pas de véritable vidéo à la demande puisqu'il faut attendre la fin du téléchargement avant de pouvoir lire le fichier vidéo. Cependant, avec l'augmentation des débits des connexions réseaux, de plus en plus de services vidéos optent pour ce système.

Les protocoles applicatifs du modèle OSI utilisés par cette méthode sont typiquement HTTP ainsi que les protocoles P2P.

### Progressive Download and Play

Le *Progressive Donwload and Play* consiste à diviser le fichier vidéo en bloc de données qui seront téléchargés et ensuite joués. Les protocoles utilisés par cette technique sont HTTP et TCP. Ces deux protocoles sont les plus répandus sur le réseau Internet, cette technique offre donc une grande couverture réseau.
Un fichier vidéo de 10 minutes sera avec cette technique par exemple découpé en 100 petits fichiers de 6 secondes chacun qui seront successivement téléchargés et joués chez le client. Cette technique est aussi appelée *Streaming HTTP* puisqu'elle utilise la méthode du *streaming* en se servant du protocole HTTP.

### Podcasting

La technique du *Podcasting* consiste à télécharger des fichiers vidéos par l'intermédiaire d'abonnements. Les auteurs publient des fichiers et c'est aux auditeurs que revient le rôle de gérer une liste de lecture avec leurs différentes souscriptions. Le téléchargement des fichiers est alors automatisé et issu des multiples sources qu'ils ont choisies. Une fois les fichiers vidéos téléchargés, l'utilisateur peut les visionner avec un simple lecteur multimédia. Cette technique est fortement employée pour la distribution d'émissions récurrentes et à destination surtout d'appareils mobiles et déconnectés temporairement d'un réseau.

Les protocoles utilisés sont le *Real Simple Syndication* (RSS) qui permet de s'abonner à différentes sources vidéos ainsi qu'HTTP pour le téléchargement.

Voici les avantages et les inconvénients des techniques de distribution vidéo. Ce tableau est issu de [7, Chapitre 11, page 154].

| **Techniques de diffusion vidéo** | **Avantages** | **Inconvénients** |
|---|---|---|
| True Streaming | Délais courts ; Supporte le contenu *live* ; Ne nécessite pas d'espace de stockage ; Supporte l'interactivité ; Diffusion *multicast* supportée | Problèmes pour traverser les pare-feux ; Il faut rester connecté au réseau en permanence |
| Download and Play | Moyen rapide d'obtenir une vidéo sur un réseau rapide ; Traverse facilement les pare-feux ; La vidéo est considérée comme un fichier normal ; | Ne supporte pas l'interactivité ; Il faut attendre la fin du téléchargement avant de pouvoir lire la vidéo ; Un espace de stockage est nécessaire ; Problèmes possibles de droits d'auteurs vu la présence du fichier vidéo chez l'utilisateur |
| Progressive Donwload and Play | La lecture peut commencer rapidement ; Traverse facilement les pare-feux ; Le fichier vidéo est mis en cache chez l'utilisateur afin de faciliter le contrôle de la lecture (avance rapide, retour en arrière) | La segmentation des fichiers vidéos peut ne pas être supportée par le lecteur vidéo ; Requiert de l'espace de stockage chez l'utilisateur |
| Podcasting | Historiquement prévu pour atteindre le public qui utilise des appareils mobiles ; Le fichier vidéo peut être lu sans être connecté au réseau ; Ne requiert pas de PCs, un simple baladeur vidéo est suffisant ; Bonne qualité audio et vidéo ; Peut être délivré automatiquement par souscription | La résolution d'écran est souvent limitée ; Ne permet pas l'interaction avec le Web. |

FIGURE 1.2 – Avantages et Inconvénients des différentes techniques de diffusion vidéo. Figure issue du livre : [7, Chapitre 11, page 154]

Dans cette section consacrée au protocoles clés de la distribution vidéo, nous devons parler à présent des différentes architectures de distribution de contenu vidéo les plus souvent rencontrées ainsi que des protocoles qui leur sont associés.

**Architectures** Avec l'usage d'Internet, le nombre d'utilisateurs potentiels pouvant se connecter à un service vidéo est devenu gigantesque. De plus, ces utilisateurs sont potentiellement distribués mondialement, c'est donc un véritable défi de distribuer de la vidéo à la demande et en particulier sur Internet. L'architecture classique Client/Serveur impose dès lors au serveur une capacité de bande passante démesurée. C'est pourquoi les vidéos à la demande sont généralement basées sur deux autres types d'architectures. Il s'agit de l'architecture *Content*

*Distribution Networks* (CDN) et *Peer-to-Peer* (P2P).

L'approche CDN a pour but de rapprocher le contenu vidéo au plus près du consommateur. Plusieurs entreprises comme Akamai [4] se sont spécialisées dans cette technique. Cette technique consiste à disposer géographiquement plusieurs serveurs à travers Internet. Ces serveurs contiendront tous une copie du contenu vidéo original. La distribution vidéo s'opère via la technique des *multicast overlay networks* (voir section 1.3.3). Youtube par exemple a opté pour ce système d'architecture. Pour plus de détails sur cette technique, consultez ce livre [2, Chapitre 7, page 618].

L'approche P2P est un système distribué où chaque client joue aussi le rôle de serveur. Dans le cas de la distribution vidéo, un client peut donc télécharger une vidéo ou une partie de celle-ci et la redistribuer à son voisin (peer). Cette approche est moins sensible à l'augmentation du nombre de clients que les architectures Client/Serveur ou CDN. L'utilisation de la bande passante étant répartie entre tous les clients et non pas uniquement entre les clients et les serveurs. L'inconvénient de cette architecture est le manque de contrôle vu son caractère décentralisé. Cela peut donc entrainer des problèmes de fiabilité et de délais non respectés.

Des architecture hybrides issues des deux architectures présentées ci-dessus ont également vu le jour. Pour plus détails sur ces architectures, veuillez vous référer au mémoire d'Y.Dony [8]

### 1.3.5 Viewing Devices

Pour regarder un contenu vidéo, plusieurs appareils sont utilisés. Les services vidéos appartenant aux catégories IPTV emploient le plus souvent des *Set-Top Boxes* (STB). Ces appareils servent d'interface entre l'équipement traditionnel des spectateurs tel qu'un écran de télévision, un enregistreur DVD ou un magnétoscope et le réseau de télévision digitale. Leur fonction est de convertir un signal IPTV en un signal compréhensible par un poste de télévision. Ils disposent généralement d'interfaces analogiques telles *Digital Visual Interface* (DVI), S-Video et *High Definition Multimedia Interface*(HDMI).

Ils fournissent également les fonctionnalités de base à l'utilisateur telles que l'accès au guide de télévision et à l'enregistrement. L'appareil est généralement fourni par la société responsable du service vidéo.

Les services vidéos appartenant aux catégories IPVOD, Internet TV et Internet Video utilisent principalement un ordinateur, un appareil mobile (GSM, PDA) ou encore tout appareil pouvant se connecter à Internet et muni d'un écran.

4. `http://www/akamai.com`

### 1.3.6 Program Choices

Le nombre de programmes ou de vidéos proposés varie fortement d'une catégorie de service vidéo à une autre. Ainsi un système IPTV distribuera en moyenne une centaine de chaînes de télévision en continu. Les fournisseurs d'accès au service IPTV effectuent une sélection de stations de télévision. Vu que ces fournisseurs payent un droit de diffusion pour chaque chaîne, ils ont tendance à sélectionner les chaînes où l'audience est relativement forte. Cette limitation à une centaine de stations de télévision s'explique aussi du fait que les systèmes IPTV sont disponibles sur réseaux privés et sur un territoire réduit. Les chaînes de télévision locales sont alors préférées.

Les systèmes Internet Video offrent, quant à eux, une quantité de vidéos plus importante que les systèmes IPTV. Ils profitent du fait que les utilisateurs fournissent la majorité du contenu vidéo et ceci de manière gratuite. De plus ces services vidéos agissent sur Internet, ils reçoivent donc potentiellement des vidéos de la planète entière.

### 1.3.7 User Experience

Un aspect à ne pas négliger est l'expérience de l'utilisateur. Ce paramètre est primordial pour les entreprises de services vidéos. Ils savent que ce paramètre est fortement lié à la satisfaction du client. Or un client satisfait a plus chance de devenir un client durable. Les entreprises prêtent donc attention pour que l'expérience que l'utilisateur devra acquérir soit minime ou que l'utilisateur puisse se raccrocher à une expérience qu'il connait déjà.

Dans cet ouvrage [7, Chapitre Introduction], une compilation de différentes études sur les habitudes des utilisateurs aux Etats-Unis au niveau des services vidéos a été faite. Cette compilation met en évidence l'évolution des habitudes des utilisateurs concernant les services vidéos pendant ces dernières années. Les spectateurs regardent de plus en plus d'émissions sur Internet (émissions sportives, YouTube, ...). Le réflexe *Internet* devient de plus en plus fréquent. L'utilisation des *Digital Video Recorder* (DVR) explose un peu partout aux Etats-Unis. Les appareils mobiles (GSM, Netbook, PDA, ...) sont également en pleine croissance et distribuent de plus en plus du contenu multimédia. Les spectateurs deviennent aussi des producteurs de vidéos, il suffit de voir le succès de YouTube pour en être fixer. L'utilisation de *podcast*, notamment pour les émissions radios, s'est aussi largement généralisée.

Pour qualifier l'expérience de l'utilisateur vis-à-vis des services vidéos, nous pouvons la comparer avec des services existants. Les systèmes IPTV ont pour objectif de remplacer la télévision analogique traditionnelle. L'utilisateur dispose d'un guide TV et peut naviguer entre les différentes chaines proposées. L'expérience de l'utilisateur avec ces systèmes ressemble donc fortement à celle qu'il avait avec la télévision câblée. Nous pouvons cependant ressentir le travail des entreprises de services vidéos, notamment à travers la publicité, pour adapter nos habitudes aux nouveaux services vidéos. Il suffit de voir la campagne de Belgacom concernant la fonctionnalité du bouton pause ou encore la campagne de Voo pour la vidéo à la demande qui remplace la location de films à la vidéothèque.

Les systèmes IPVOD ajoutent aux systèmes IPTV des fonctionnalités de vidéo à la demande et d'enregistrement. L'expérience de l'utilisateur ressemble alors à celle qu'il avait avec

les appareils d'enregistrements classiques (magnétoscope, enregistreur DVD).

Le systèmes *Internet TV* et *Internet Video* sont disponibles sur Internet. L'expérience de l'utilisateur avec ces systèmes ressemble donc à la navigation sur Internet. L'utilisateur navigue au sein d'un site web dans un catalogue de vidéos, sélectionne une vidéo et la regarde sur son ordinateur.

### 1.3.8 Channel Change Time

Le *Channel Change Time* est le temps nécessaire pour qu'un service vidéo commence la lecture de la vidéo une fois que l'utilisateur en a fait la demande. La plupart des services vidéos utilise la méthode d'*unicasting* (voir section 1.3.3). Dès lors, dès qu'un utilisateur change de programme, le service vidéo et le réseau doivent réagir pour envoyer la nouvelle chaine demandée. Ceci est très différent de la télévision traditionnelle où toutes les programmes sont envoyés en parallèle. Dans un tel système, lorsque l'utilisateur change de programme, l'écran de télévision doit simplement changer la fréquence du canal regardé.

Les systèmes IPTV essaient de remplacer la télévision câblée traditionnelle. Comme vu dans la section 1.3.3, ces services vidéos utilisent le *multicast*. Il faut donc compter des délais pour quitter un groupe *multicast* diffusant un programme télévisé et en rejoindre un autre. Les délais pour changer de programme peuvent être longs. Pourtant nous remarquons que les temps de réaction de ces systèmes sont de l'ordre de 1 à 2 secondes ce qui est tout à fait raisonnable. En fait ils utilisent différentes techniques pour éviter des délais trop longs. Un flux *unicast* peut être employé dès qu'un changement de programme survient, le temps que la jonction au nouveau groupe *multicast* soit effective. La diffusion *multicast* peut aussi envoyer les chaînes adjacentes. Pour plus de détails sur ces techniques, référez-vous à Huseyin Uzunalioglu [6]

.

Les systèmes Internet Video ont des temps de réaction plus grands. Ceci est du à l'emploi de navigateur Web et de *buffers*. La technique employée par ces services vidéos est le *Progressive Download And Play* (voir 1.3.4). Il faut donc attendre que les *buffers* soit vidés et rechargés avec la nouvelle vidéo avant que la lecture ne puisse commencer.

### 1.3.9 Rewind / Fast-Forward

Un argument en faveur de la télévision numérique est la possibilité d'interagir avec une vidéo. Les fonctionnalités sur le contrôle de la lecture, notamment le retour en arrière et l'avance rapide, sont appréciés des utilisateurs.

Pour pouvoir interagir au sein d'une vidéo, il faut que le contenu de la vidéo ait été enregistré au préalable. Les fonctionnalités de contrôle de lecture d'une vidéo sont directement liées au *Content Types* (voir 1.3.11).

Les systèmes IPTV et Internet TV diffusent en temps réel la vidéo. Ils ne permettent donc pas d'interagir avec la vidéo. L'utilisateur sélectionne simplement une chaine et la regarde en temps réel. C'est pourquoi les systèmes IPTV et Internet TV sont souvent accompagnés d'un système d'enregistrement. Un programme enregistré possèdera alors des fonctionnalités sur le

contrôle de la lecture.

Le systèmes IPVOD et Internet Video travaillent essentiellement avec du contenu vidéo pré-enregistré. Les fonctionnalités de contrôle de lecture, notamment l'avance rapide et le retour en arrière qui nous intéressent ici, sont alors disponibles.

Les fonctionnalités de contrôle de lecture sont gérées par le protocole *Real Time Streaming Protocol* (RTSP) qui permet l'échange de commandes telles que pause, avance rapide et retour en arrière entre le serveur et le client. Pour plus de détails sur ce protocole, référez-vous à : [2, Chapitre 7, page 604].

### 1.3.10 Production Values

La valeur de la production d'une vidéo représente le coût de préparation d'une vidéo. Les coûts de préparation de ce contenu vidéo incluent le montage de la vidéo, l'encodage dans un format vidéo supporté par les systèmes IPTV, ....

Les services vidéos peuvent être classifiés selon la valeur de la production vidéo. Cette production vidéo peut être réalisée par des professionnels ou par des utilisateurs amateurs. Le contenu produit professionnellement offre de manière générale une meilleure qualité vidéo que le contenu vidéo généré par des utilisateurs amateurs.

Les services vidéos proposant un contenu vidéo produit de manière professionnelle sont généralement payants. Ils répercutent sur les utilisateurs le coût de préparation des vidéos ainsi que des droits d'auteurs. Les systèmes IPTV et IPVOD et Internet TV proposent du contenu vidéo produit professionnellement.

Contrairement aux systèmes ci-dessus, l'Internet Video utilise principalement du contenu vidéo généré par les utilisateurs. La préparation du contenu vidéo est réalisée par l'utilisateur, ce qui permet au service vidéo de rendre ces vidéos gratuites. La qualité de ces contenus vidéos est plus faible vu que ce sont des amateurs qui réalisent la plupart du temps la production et non des professionnels.

### 1.3.11 Content Types

Les services vidéos peuvent faire le choix d'enregistrer leurs contenus vidéos (*pre-recorded* ou de les distribuer directement en temps réel (*Live*). La distribution en temps réel se prête bien aux événements sportifs ou aux actualités qui doivent être transmises en temps réel. Dans ce cas le contenu vidéo n'est pas enregistré et est diffusé directement par le réseau. Les systèmes IPTV et Internet TV peuvent recourir à ce système contrairement aux catégories IPVOD et Internet Video où le contenu doit être obligatoirement enregistré. Le fichier n'étant pas enregistré, les fonctionnalités de contrôle de lecture tel que l'avance rapide ou le retour en arrière ne sont pas possibles (voir 1.3.9).

L'alternative consiste à enregistrer au préalable le contenu vidéo. Les fonctionnalités de contrôle de lecture sont alors disponibles. Le fait que cette technique autorise le contrôle de la lecture et est supportée par les quatre catégories de services vidéos en font la technique la plus répandue. Cependant il faut être conscient que cette technique nécessite de l'espace de

stockage assez conséquent sur les serveurs des services vidéos.

Le stockage permet d'aborder un aspect important des services vidéos qui est la compression. Le succès même d'un service multimédia peut dépendre de la méthode de compression. Un exemple de ceci est le format MP3. Sans cette méthode de compression l'avènement des lecteurs MP3 n'aurait pas eu lieu. C'est également grâce à la compression que les vidéos en haute définition peuvent circuler, un signal vidéo HD non compressé occupe 1,5 Gbps ce qui représente 1000 fois la capacité de nos lignes ADSL actuelles. La compression vidéo est donc nécessaire pour épargner le stockage, la bande passante et faciliter le transport. Plusieurs protocoles existent : MPEG1, MPEG2, MPEG4, H.261. Pour plus de détails sur la compression vidéo, veuillez vous référer à [2, Chapitre 7, page 596] et [7, Chapitre 6].

### 1.3.12 Program Library

La librairie de contenu vidéos disponibles peut être constituée par les fournisseurs de services vidéos. Dans ce cas, on parle de *Walled garden*, il s'agit d'une collection de contenu vidéo sélectionnée et contrôlée par le fournisseur de services vidéos.

Le systèmes IPTV et IPVOD ont pour la plupart un catalogue de vidéos du type *Walled Garden.*

Les systèmes Internet TV et Internet Video constituent leur librairie de contenu vidéos à partir des utilisateurs. L'utilisateur participe donc à l'élaboration du catalogue vidéo. La qualité du contenu vidéo varie fortement suivant les compétences et les motivations de l'utilisateur.

### 1.3.13 Ownerships Rights

Les droits d'auteurs ainsi que la sécurité sont deux paramètres importants dans le monde des services vidéos. Un fournisseur de services vidéos doit garantir aux propriétaires de vidéos professionnelles que celles-ci ne seront pas piratées ou copiées illégalement. Les fournisseurs de services vidéos doivent aussi vérifier les accès à leur réseau de distribution et se protéger contre les accès non autorisés.

Nous pouvons distinguer deux ensembles de systèmes couramment employés à savoir le *Conditional Access* et les *Digital Right Management* (DRM). Tout d'abord on distingue le *Conditional Access* responsable de la gestion des accès. Son but est essentiellement de gérer ce qu'un utilisateur a le droit de voir. Le système de *Conditional Access* se décline sous plusieurs formes. Nous retrouvons dans le cadre des services vidéos IPTV des *Smart Card* implantées directement dans les appareils STB (voir section 1.3.5) et qui contiennent la clé privée permettant de décrypter les contenus vidéos. Pour les services vidéos sur Internet, la sécurité est moindre et nous retrouvons les méthodes traditionnelles d'authentification telles que le login et le mot de passe.

Un deuxième système est utilisé, il s'agit des techniques de *Digital Right Management* (DRM) dont l'objectif est de gérer ce qu'un utilisateur peut faire avec un contenu vidéo. Le systèmes DRM aident les fournisseurs d'accès vidéo à garantir que les droits d'auteurs ne seront pas violés auprès des producteurs de vidéos. Ils s'occupent principalement de vérifier

le nombre de fois qu'un utilisateur peut visionner une vidéo, l'horaire à laquelle cet utilisateur peut regarder une vidéo, les copies d'une vidéo qui sont effectuées. Les techniques DRM utilisent des méthodes de sécurité déjà existantes telles que le chiffrement, le *scrambling*, le *watermarking*. Pour plus de détails sur ces techniques, référez-vous à [7, Chapitre 7].

Les systèmes IPTV et IPVOD utilisent des mécanismes de protection des droits d'auteurs assez poussés tandis que les systèmes Internet TV et Internet Video possèdent une protection plus faible voire inexistante.

### 1.3.14 Revenues Models

L'essor de la vidéo sur IP peut s'expliquer en partie par le fait qu'elle propose aux entreprises une grande variété de modèles de revenus. L'avènement de l'utilisation du protocole IP à la place de solutions propriétaires a permis à de nouvelles entreprises de se lancer sur ce marché.

Parmi les modèles les plus souvent employés, on peut citer

1. l'abonnement à un ensemble de chaînes de télévision
2. le paiement par épisode de la vidéo à la demande
3. les offres d'abonnement des systèmes avec enregistrements
4. les offres commerciales groupées (Téléphone, Internet, télévision)
5. les revenus publicitaires des systèmes de contenu généré par l'utilisateur

L'accès à la technologie IP, grâce aux fournisseurs d'accès et aux nombreux systèmes autonomes est mondial. Dès lors l'inconvénient majeur pour les entreprises est le nombre de concurrents potentiels. De plus autour du multimédia s'est développé un esprit de gratuité, d'illégalité. C'est donc un véritable défi pour une société de proposer un service viable économiquement.

### 1.3.15 Examples Providers

La liste de catégorie de services vidéo propose des exemples concrets pour chaque catégorie. Cependant il s'agit d'exemples américains. Voici quelques exemples concrets européens.

Le service de base de Belgacom TV est un exemple d'IPTV. Ce service vidéo propose un ensemble de chaînes de télévision et un guide TV. Les clients sont connectés sur le réseau privé de Belgacom, Belgacom peut ainsi gérer la qualité de service et implémenter le *multicasting*. Belgacom loue à ces clients un STB capable de connecter leur écran de télévision au réseau de télévision digitale. Les clients payent un abonnement à Belgacom.

Un exemple de la catégorie IPVOD est le site de la FNAC [5] qui propose de la vidéo à la demande. Des films peuvent être loués et vus pour une durée de 48 heures moyennant un paiement. Ce service vidéo est disponible par Internet, utilise le *Progressive Download and*

5. `http://www.vod.fnac.com`

*Play* et une protection DRM.

La RTBF [6] propose ses chaînes en direct sur Internet. Il s'agit d'un bon exemple pour la catégorie Internet TV.

La catégorie Internet Video est, quant à elle, bien représentée par les sites de YouTube [7] et de Facebook [8]. Ces sites proposent gratuitement un large catalogue de vidéos essentiellement produites par les internautes.

## 1.4 Description du système de distribution de contenu vidéo étudié

Dans cette section nous présentons le service vidéo implémenté par le simulateur.

Ces dernières années, notamment grâce à l'emploi de nouvelles technologies d'accès réseau, Internet s'est répandu dans plus en plus de foyers. Ces technologies d'accès réseau, telles que la fibre optique par exemple, ont permis d'avoir des connexions réseaux à plus haut débit permettant le transport de gros volumes de données. Dès lors Internet est devenu un support de choix pour la distribution vidéo. Ce support a permis également de se détacher de la télévision traditionnelle, où les émissions télévisées sont regardées en temps réel, pour créer de plus en plus de services de vidéos à la demande. Beaucoup de services vidéos basés sur la technologie IP sont alors apparus avec des formules très diverses. Nous pouvons les classer dans différentes catégories comme nous l'avons vu dans la section précédente (voir 1.3).

Le service vidéo implémenté par le simulateur est un *Network-based TV recorder*. Il s'agit d'un service vidéo proposant à ces clients d'enregistrer et de télécharger en ligne des programmes de télévision. L'accès se fait via un site Web. Chaque utilisateur enregistré a la possibilité via un *Electronic program guide* (EPG) de sélectionner les programmes qu'il désire enregistrer. Les enregistrements sont stockés directement sur les serveurs de l'entreprise proposant ce service vidéo. Ensuite, l'utilisateur peut télécharger les vidéos enregistrées au préalable soit via le protocole HTTP ou encore via les protocoles d'échange de fichier P2P. Cependant la majorité des utilisateurs téléchargent via la solution HTTP, c'est pourquoi le simulateur se concentrera sur ce type de transfert.

Un exemple de ce type de service vidéo est *OnlineTVRecorder* [9].

Le simulateur s'intéresse au temps de téléchargement des clients. Il simule donc la partie transfert entre le serveur du service vidéo et les clients. Il ne s'occupe pas de la partie enregistrements des programmes télévisés.

Plus de précisions sont données dans l'article à la base de l'étude [5].

6. `http://www.gowatv.com/rtbfsat-direct-tv-streaming.php`
7. `http://www.youtube.fr`
8. `http://fr-fr.facebook.com/`
9. `http://www.onlinetvrecorder.com`

## 1.5 Catégorie du service vidéo implémenté par le simulateur

Dans cette section, nous discutons des différents critères de la liste (voir 1.3) pour le service vidéo du simulateur.

La catégorie la plus proche du service vidéo implémenté par le simulateur est IPVOD. La différence fondamentale entre la catégorie IPVOD et le service vidéo du simulateur réside dans le fait que le service vidéo implémenté par le simulateur enregistre des émissions télévisées. Il n'est donc pas possible de regarder instantanément une vidéo contrairement à la catégorie IPVOD. Nous allons analyser le service vidéo suivant les critères de la liste des catégories (voir section 1.3).

**Network Type & Quality of Service** Le type de réseau du service vidéo est clairement public puisqu'il s'agit d'un service présent sur Internet. Nous avons vu que la qualité de service, en terme de réseau informatique, est fortement liée au type de réseau. Il n'y a pas d'exception ici, vu que le réseau est Internet, le service vidéo ne peut pas fixer des règles pour gérer la qualité de service. Il doit s'en accommoder.

**Multipoint Method** Nous rappelons que nous nous focalisons sur le téléchargement des vidéos enregistrées par les utilisateurs et non pas par le système d'enregistrement des programmes télévisés. La méthode utilisée par le service vidéo est donc l'*unicasting*. Le téléchargement s'effectuant par HTTP directement entre un client et le serveur du service vidéo.

**Key Protocols** Le protocole utilisé par le service vidéo implémenté par le simulateur est HTTP. La technique de distribution vidéo utilisée est le *Download and Play*. Les fichiers vidéos sont entièrement téléchargés avant d'être regardés. L'architecture classique client/serveur est utilisée par le service vidéo. Le service vidéo s'écarte donc pour ce critère de la catégorie IPVOD qui utilise le *Progressive Download And Play*.

**Viewing Devices** Le service vidéo du simulateur utilise un portail Web comme support pour ces utilisateurs. Les utilisateurs doivent donc disposer d'un appareil capable de se connecter à Internet et d'en afficher le contenu. Nous pouvons supposer que la majorité des utilisateurs emploie un ordinateur pour regarder les vidéos. Cependant comme les utilisateurs récupèrent les vidéos sur leur ordinateur, ils peuvent ensuite regarder ces vidéos sur n'importe quel appareil pouvant lire le format du fichier vidéo même si cet appareil n'est pas connecté à Internet. On peut donc imaginer que les vidéos soient regardées depuis un appareil mobile, depuis un lecteur DVD de salon. Ce service vidéo offre donc une grande souplesse à ce niveau.

**Program Choices** Si nous regardons le site *OnlineTVRecorder*[10], nous pouvons remarquer que ce site propose des chaines allemandes, américaines, anglaises, .... Le nombre de chaînes disponibles correspond donc bien à un millier de chaînes. Contrairement à la distribution IPTV qui est locale et où le nombre de chaînes est plus limité, ce service peut offrir un large

10. `http://www.onlinetvrecorder.com`

catalogue. Chaque chaîne de télévision présente sur Internet peut potentiellement entrer dans ce catalogue.

**User Experience** L'expérience de l'utilisateur vis-à-vis de ce service vidéo est similaire à la navigation sur Internet mais aussi à l'emploi de DVR. Ce service étant disponible par le biais d'un site Web, l'utilisateur retrouvera l'expérience qu'il a acquise dans la navigation Internet. Le site propose l'enregistrement d'émissions de télévision et de programmer l'enregistrement de celles-ci. L'utilisateur retrouve alors l'expérience qu'il a acquise avec les appareils d'enregistrements classiques(magnétoscope, enregistreur cassette, enregistreur DVD, ...).

**Channel Change Time** Ce critère n'est pas applicable ici vu que le service vidéo ne propose pas de regarder les chaînes en temps réel mais bien de les enregistrer.

**Rewind / Fast-forward & Content Types** La catégorie IPVOD ne propose que du contenu pré-enregistré. Nous avons vu que pour permettre des fonctionnalités de contrôle de lecture, il faut impérativement que le contenu soit enregistré. Pour le service vidéo du simulateur, c'est bien évidemment le cas puisque le fichier vidéo est entièrement téléchargé avant sa lecture. Nous pouvons dès lors avancer, revenir en arrière au sein de la vidéo.

**Production Values** Le service vidéo implémenté par le simulateur possède un contenu de type professionnel. En effet le service vidéo agrège les programmes télévisés de chaînes professionnelles.

**Program Library** Le service vidéo du simulateur propose un contenu vidéo provenant de chaînes de télévision professionnelles présentes dans le monde entier. Cependant les responsables du service vidéo sélectionnent les chaînes qui feront partie de leur catalogue. Nous pouvons donc dire que pour ce critère le service vidéo est *Walled Garden* tout comme la catégorie IPVOD.

**Ownership Rights** Le service vidéo OnlineTVRecorder [11] utilise un logiciel d'encryption que les utilisateurs installent afin de pouvoir décrypter les fichiers vidéos téléchargés. Ce logiciel gère les accès aux fichiers ainsi que le nombre de fois qu'une vidéo peut être décodée. Il s'agit donc typiquement d'un logiciel utilisant des techniques DRM.

**Revenue Models** Le service vidéo du simulateur s'appuie sur plusieurs modèles de revenus. Les revenus publicitaires sont exploités, un mode de paiement où l'utilisateur clique sur des bannières publicitaires a été mis en œuvre. Le formules de paiement à la carte et d'abonnement sont également présentes pour ce service vidéo.

11. `http://www.onlinetvrecorder.com`

## 1.6 Conclusion

Dans ce chapitre, nous avons présenté une classification des systèmes de distribution de contenu vidéo en quatre catégories. Une liste de critères techniques, économiques et culturels permettant le classement dans l'une de ces catégories a été expliquée. Cette classification et cette liste de critères ne sont bien évidemment pas exhaustives mais elles permettent néanmoins de différencier les services vidéos présents actuellement sur le marché. Nous avons ensuite décrit le service vidéo implémenté par le simulateur. Finalement, après avoir analysé le service vidéo du simulateur en regard de la liste de critères, nous avons classé ce service vidéo dans la catégorie IPVOD.

# Chapitre 2

# Introduction à la simulation par événements discrets

## Sommaire


**2.1 Classification des modèles de simulation à événements discrets . 21**
**2.2 Terminologie de la simulation par événements discrets . . . . . 23**
**2.3 Démarche générale d'une simulation par événements discrets : présentation des éléments essentiels . . . . . 25**
2.3.1 La génération des données d'entrée . . . . . 25
2.3.2 La simulation . . . . . 28
2.3.3 Analyse des résultats . . . . . 29


L'implémentation du simulateur est réalisée selon l'approche de la simulation par événements discrets. C'est pourquoi ce chapitre introduit de manière générale la simulation par événements discrets. Les éléments théoriques essentiels et la terminologie de cette discipline y sont présentés. Nous faisons référence à ces éléments dans la suite de document.

## 2.1 Classification des modèles de simulation à événements discrets

Cette section a pour objectif de situer les modèles de simulation par événements discrets parmi des différentes catégories de modélisation de systèmes existants.

La simulation par événement discret consiste en différentes techniques mathématiques afin de modéliser, simuler et analyser les performances d'un système. La simulation par événement discret s'emploie typiquement dans le cadre de systèmes où d'autres types de modélisation sont rendues difficiles par le nombre de variables aléatoires du système. Pour effectuer une simulation par événements discrets, nous devons tout d'abord élaborer un modèle de simulation.

Nous présentons dans cette section les différents modèles de systèmes existants. Ceci afin de montrer à quelle catégorie appartiennent les modèles de simulation par événements discrets.

Il existe une taxonomie des types de modèles de systèmes illustrée ci-dessous.

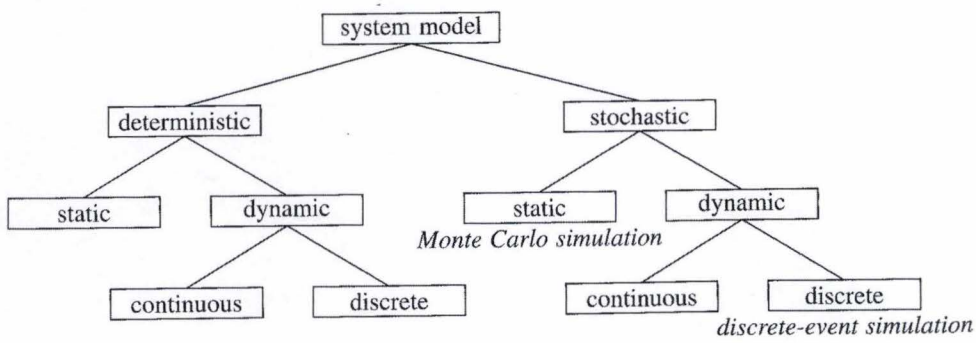



FIGURE 2.1 – Types de modèles de systèmes. Figure issue de [3, Chapitre1, page 2]

Comme nous le voyons sur le schéma ci-dessus, la simulation par événement discret permet de simuler le modèle d'un système de manière stochastique, où le temps est discrétisé.

Un système stochastique par opposition à un système déterministe contient une ou plusieurs composantes aléatoires. Beaucoup de systèmes possèdent cette propriété. Par exemple, une machine industrielle peut tomber en panne, les requêtes sur un site web arrivent de manière aléatoire. L'avantage de la simulation par événements discrets est de pouvoir modéliser ces comportements aléatoires de manière assez simple.

Un système dynamique par opposition à un système statique est un système où la variable du temps possède une importance. L'évolution du temps a une importance sur l'état des variables du systèmes. Par exemple, si nous nous intéressons à la probabilité d'avoir un gagnant à la loterie, ceci peut être modéliser de manière statique. Ce modèle est valable quel que soit le période de l'année. Il n'y a aucune dépendance vis-à-vis du temps. Par contre si nous nous intéressons à la probabilité d'avoir un gagnant à la loterie pendant les quatre dernières semaines, un système dynamique doit être utilisé.

Un système dynamique se décline en deux classes que sont les systèmes continus et les systèmes discrets. Le systèmes continus possèdent des variables qui évoluent de manière continue dans le temps. Un système discret, par opposition, a un état dépendant du temps. L'état des variables du système change en fonction d'événements qui surviennent dans le temps. Par exemple, le nombre de travaux dans une file d'attente d'impression varie en fonction du temps. Ce nombre évolue de manière discrète. Dès qu'un travail arrive, le nombre de travaux de la file d'attente est incrémenté et il est décrémenté lorsqu'un travail est terminé.

## 2.2 Terminologie de la simulation par événements discrets

Dans cette section nous introduisons la terminologie utilisée par la simulation par événements discrets. A cette fin, nous parlons brièvement de la théorie des files d'attentes sur laquelle s'appuie la simulation par événements discrets. Nous illustrons cette terminologie via un exemple.

**Théorie des files d'attente** La théorie des files d'attente est une théorie mathématique relevant du domaine des probabilités, qui étudie les solutions optimales de gestion des files d'attente. Elle peut s'appliquer à différentes situations telle que la gestion des requêtes HTTP pour un serveur Web ou encore la gestion des files d'attentes aux guichets de la poste.

Voici un exemple (Figure 2.2) de modélisation conceptuel d'une file d'attente simple.

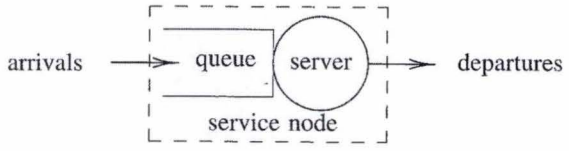



FIGURE 2.2 – Modèle conceptuel d'une file d'attente simple. Figure issue de [3, Chapitre 1, page 11]

Expliquons maintenant les différents composants de ce schéma ainsi que leurs terminologies.

**Les clients** Dans la théorie des files d'attente, nous parlons de clients. Il s'agit d'éléments qui entrent dans le système, sont pris en charge par un serveur et sortent du système. Suivant le système modélisé, ces clients peuvent représenter des personnes ou des travaux d'impression par exemple.

**Les arrivées** Dans la théorie des files d'attente, le système est caractérisé par des instants d'arrivée de clients. Ceux-ci sont modélisés par une loi de probabilité qui décrit le comportement de ces arrivées. Le processus stochastique décrivant ces instants d'arrivées est habituellement le processus de Poisson. Celui-ci décrit finalement un processus de comptage d'occurrences sur base de variables aléatoires indépendantes qui suivent une même loi exponentielle. Nous nous en référons à [3] pour ces lois de probabilité.

**La file d'attente (queue)** Un client, une fois arrivé, est placé dans une file d'attente avant d'être servi. La file d'attente possède une discipline qui régit la sélection du prochain client qui sera servi. Les disciplines des files d'attente les plus courantes sont :

- *Firt In, First Out* (FIFO)
- *Last In, First Out* (LIFO)

- *Service In Random Order* (SIRO)
- un mécanisme de priorité

Une file d'attente possède également une capacité qui peut être fixe ou illimitée.

**Le serveur** Dans la théorie des files d'attente, le serveur représente l'unité de traitement d'un client. Dès qu'un client entre en service, il est pris en charge par ce serveur. Chaque client possède un temps de service qui obéit à une loi de probabilité tout comme les arrivées. Le serveur peut être unique ou multiple.

Le serveur définit la politique de traitement des clients. Diverses stratégies sont envisageables. Nous pouvons citer le partage du serveur entre tous les clients de manière égalitaire(*processor sharing*) ou une politique de priorité donnée à certains clients.

**Les départs** Les départs de clients modélisent la fin du service. C'est lors de cette phase, notamment, que des observations statistiques sont collectées telles que le temps de séjour du client par exemple.

**Notation de Kendall** Il existe un formalisme qui décrit un modèle de file d'attente suivant les paramètres expliqués ci-dessus. Il s'agit de la notation de *Kendall*. Cette notation se compose de six symboles a/s/C/K/m/Z dont voici la signification :

- a : indique la loi de probabilité des arrivées,
- s : indique la loi de probabilité des temps de services,
- C : indique le nombre de serveurs,
- K : indique la capacité totale du système, c'est-à-dire le nombre maximum de clients dans la file d'attente et en service.
- m : indique la population totale des clients,
- Z : la discipline de la file d'attente,

**Mesures de performance des files d'attente** L'objectif de la modélisation d'une file d'attente est la mesure des performances de cette file d'attente. Pour ce faire, différentes variables telles que le temps total que passera un client dans le système ou encore le nombre de clients dans la file d'attente sont calculés. La théorie des files d'attente définit donc plusieurs variables couramment manipulées dont voici les définitions :

- le temps d'attente : il s'agit du temps d'attente d'un client à l'intérieur de la file d'attente
- le temps de service : il s'agit du temps nécessaire au serveur pour servir un client
- le temps de séjour : il s'agit du temps total que passera un client dans le système. Il s'agit donc du temps d'attente ajouté au temps de service du client.
- le nombre de clients dans le système
- le nombre de clients dans la file d'attente

Pour ces différentes variables, nous sommes intéressés par leurs distributions ainsi que leur moyenne et leur écart-type pour l'ensemble des clients.

## 2.3 Démarche générale d'une simulation par événements discrets : présentation des éléments essentiels

Dans cette section, nous voyons la démarche générale pour réaliser une simulation par événements discrets. Celle-ci va nous permettre d'aborder les éléments essentiels à la réalisation d'une simulation par événements discrets.

La simulation par événements discrets se décompose en trois étapes principales :

- la génération des données d'entrée
- la simulation
- l'analyse des résultats

### 2.3.1 La génération des données d'entrée

La simulation par événements discrets s'appuie sur la théorie des files d'attente (voir 2.2). Le système étudié est donc notamment décrit par des arrivées et des temps de service. Ces informations constituent des données d'entrée nécessaires pour la simulation. Le simulateur doit habituellement connaitre au moins l'instant d'arrivée du client ainsi que le temps de service nécessaire pour le client.

Une première approche consiste à connaitre à l'avance toutes ces informations nécessaires avant de démarrer la simulation. Nous pouvons alors imaginer un fichier avec toutes les données d'entrées nécessaires qui sera utilisé par la simulation en vue de produire des résultats.

Une deuxième approche consiste à générer ces informations à la demande au fur et à mesure de la progression de la simulation.

Cependant, quelle que soit l'approche, un mécanisme pour générer ces données d'entrée s'avère nécessaire. Nous savons également que ces données d'entrées obéissent à des lois de probabilités (voir 2.2) et qu'elles doivent posséder un caractère aléatoire. C'est pourquoi nous introduisons ici les générateurs de nombres aléatoires qui permettent à la simulation par événements discrets d'obtenir une valeur aléatoire. Ensuite nous présentons la fonction inverse de distribution issue de la discipline des statistiques. Celle-ci est responsable dans la simulation par événements discrets de traduire une valeur aléatoire en une valeur aléatoire respectant une loi de probabilité. De cette manière, la simulation par événements discrets est capable de générer les temps d'arrivées et de services des clients.

**Les générateurs de nombres aléatoires** Les générateurs de nombres aléatoires sont essentiels pour effectuer des simulations par événements discrets de manière correcte. Ils sont

responsables notamment de générer les temps d'arrivée et de service d'un système de files d'attente (voir section 2.2). Un générateur de nombres aléatoires se définit par une fonction qui renvoie un nombre compris entre 0.0 et 1.0 tel que dans cet intervalle chaque valeur est susceptible d'être choisie de manière égale. Les valeurs renvoyées par un générateur de nombre aléatoire doivent donc être statistiquement indiscernables.

Les générateurs de nombres aléatoires utilisés par la simulation par événements discrets sont des générateurs de nombres pseudo-aléatoires. Cela signifie que les nombres générés sont suffisamment indépendants les uns des autres mais que la sortie de ces nombres n'est pas aléatoire. En effet, ces générateurs sont exécutés sur des ordinateurs qui sont, quant à eux, déterministes. Il en résulte que la séquence de nombres aléatoires générée est reproductible. Le générateur peut retrouver après un certain temps les mêmes conditions et donc renvoyer une valeur aléatoire déjà sortie auparavant. Ce phénomène s'appelle la période d'un générateur et nous devons y prendre garde. Il existe différents algorithmes pour réaliser un générateur de nombres pseudo-aléatoires. Nous nous concentrons sur l'algorithme de *Lehmer*, aussi appelé algorithme linéaire congruentiel. Cet algorithme est le plus utilisé en simulation par événements discrets.

Nous donnons ci-dessous une définition de l'algorithme de *Lehmer* à titre indicatif ainsi qu'une explication sur la période d'un générateur de nombres aléatoires. Cette définition est issue de [3, Chapitre 2, page 39].

**Définition de l'algorithme de *Lehmer* 1.**

Les générateurs de nombres pseudo-aléatoires sont construits sur l'algorithme de *Lehmer*. Cet algorithme est caractérisé par deux paramètres :

- *un modulo* $m$ : un grand nombre premier qui est fixé
- *un multiplicateur* $a$ : un entier fixé dans l'ensemble $\mathbb{X}_m$

et la génération de la séquence d'entiers $x_0, x_1, x_2, ...$ via l'équation itérative suivante

$$x_{i+1} = g(x_i) \quad i = 0, 1, 2, ...$$

où la fonction $g(.)$ est définie pour tous les $x \in \mathbb{X}_m = \{1, 2, ..., m-1\}$ telle que $g(x) = ax \ mod \ m$

et la semence(*seed*) de départ $x_0$ est choisie dans l'ensemble $\mathbb{X}_m$. Les générateurs basés sur cet algorithme sont appelés générateur de *Lehmer* ou encore générateur congruentiel linéaire.

La séquence produite par un générateur de nombres pseudo-aléatoires basé sur l'algorithme de *Lehmer* possède une période. Cette période représente un nombre à partir duquel la séquence de nombres aléatoires va se répéter. Elle est directement liée aux modulo $m$ et au multiplicateur $a$ employés. Cette période est donc un facteur important car il faut éviter à tout prix que le générateur de nombres aléatoires renvoie à nouveau les mêmes valeurs pour une même simulation. C'est pourquoi le choix du modulo $m$ et du multiplicateur $a$ doit être

bien étudié afin d'avoir la période la plus longue possible. Ainsi les risques de répétition seront évités. Nous nous en référons à [3, Chapitre 2, page 40].

**Génération de variables aléatoires** Comme nous l'avons vu, un générateur de nombres aléatoires est capable de renvoyer une valeur aléatoire comprise entre 0 et 1. Cependant les temps d'arrivées et de services que nous voulons distribuer aléatoirement doivent respecter une loi de probabilité. C'est pourquoi nous utilisons la fonction de distribution inverse (FDI) qui permet à partir d'un nombre aléatoire de retourner une valeur située sur la fonction de distribution d'une loi de probabilité. La fonction de distribution inverse est liée à la fonction de distribution cumulative, c'est pourquoi nous donnons ci-dessous la définition de la fonction de distribution cumulative (FDC). La fonction de distribution cumulative représentent la probabilité qu'une valeur soit trouvée à une valeur plus petite ou égale à une valeur donnée. Ces deux fonctions seront couramment employées par le simulateur. Voici leurs définitions :

**Définition de la fonction de distribution cumulative (FDC) 2.**

La fonction de distribution cumulative (FDC) d'une variable aléatoire discrète $X$ est une fonction $F(.)$ définie pour chaque $x \in \mathbb{X}$ tel que

$$F(x) = Pr(X <= x) = \sum_{t<=x} f(t)$$

où la somme couvre toutes les valeurs de $t \in \mathbb{X}$ pour lequel $t <= x$.

**Définition de la fonction de distribution inverse (FDI) 3.**

Soit X une variable aléatoire discrète avec FDC F(.). La fonction de distribution inverse de X est la fonction F* :(0,1) -> $\mathbb{X}$ défini pour toutes les valeurs de u $\in$ (0,1) tel que

$$F*(u) = min_x\{x : u < F(x)\}$$

où le minimum est toutes les valeurs possibles de $x \in \mathbb{X}$. Cela signifie que, si $F^*(u) = x$, alors $x$ est la valeur la plus petite possible de $\mathbb{X}$ pour lequel $F(x)$ est plus grand que $u$.

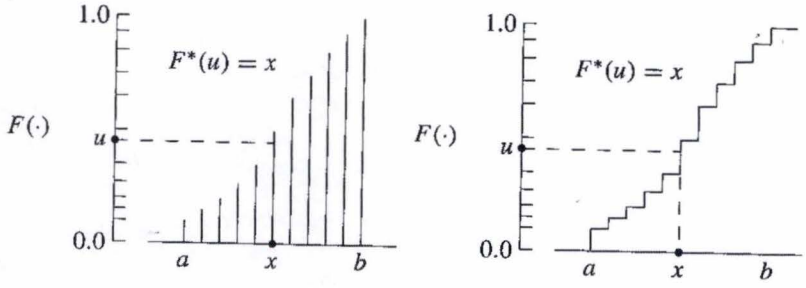



FIGURE 2.3 – Génération d'une variable aléatoire discrète. Figure issue de [3, Chapitre 6, page 237]

Nous voyons sur la figure 2.3 la traduction d'une valeur aléatoire $u$ en une valeur aléatoire appartenant à une loi de probabilité. Sur la figure 2.3 représentant la fonction de distribution cumulative d'une loi exponentielle, un nombre aléatoire $u$ d'une valeur de 0.45 a été généré. Une droite est tracée sur l'axe des ordonnées à la hauteur de $u$. Dès que la droite touche la fonction de distribution cumulative, la valeur correspondante sur l'axe des abscisses est prise. C'est cette valeur qui sera la valeur aléatoire générée et qui suit une loi de probabilité.

Cette traduction ne peut pas s'opérer avec toutes les distributions notamment avec l'Erlang qui est utilisée dans la dernière partie de ce travail. Une autre méthode est alors employée pour réaliser l'inversion, celle-ci sera expliquée dans le chapitre consacré à l'Erlang.

### 2.3.2 La simulation

La simulation en elle-même constitue la deuxième étape lors d'une simulation par événements discrets. Durant cette phase, le comportement du système étudié est simulé et des observations sont collectées. Il faut toutefois exécuter la simulation pendant un laps de temps assez long afin d'avoir une vue représentative du comportement général du système. La simulation par événements discrets parle d'état stationnaire. Nous introduisons cette notion ci-dessous.

**l'état stationnaire** Une simulation par événements discrets doit s'exécuter un certain temps avant d'arriver à un état stationnaire c'est-à-dire un état proche du comportement général du système. Intuitivement, nous pouvons comparer ceci au comportement d'une file d'attente à la poste. A l'heure d'ouverture le système se met en marche et progressivement la file d'attente atteint un régime de croisière. Si nous voulons donc tirer des résultats sur le comportement de cette file d'attente de manière générale, nous devons répéter l'expérience pendant plusieurs jours sans discontinuité afin d'avoir des données représentatives de la situation générale. Si nous nous contentons d'analyser une courte période, nous risquons d'être dans la phase de lancement ou une phase particulièrement creuse et non dans l'état stationnaire. Il faut donc

s'assurer que la simulation ait atteint un régime stationnaire avant de pouvoir tirer des résultats sur le comportement général d'un système.

Différents outils peuvent nous aider à s'assurer que nous avons atteint un régime stationnaire. Le nombre d'observations collectées est une première indication. Si ce nombre est trop faible, la simulation n'a sûrement pas duré assez longtemps. Il faut savoir que les résultats sont analysés par la suite par des outils statistiques. Les statistiques ont besoin d'un nombre minimum d'observations pour pouvoir tirer des résultats valables.

L'expérimentation permet aussi de détecter si nous nous approchons de l'état stationnaire. Nous pouvons lancer différentes simulations avec des temps différents et ensuite les comparer. Lorsque les observations sont proches et se recoupent, cela permet d'avoir une indication sur le temps minimum nécessaire pour que la simulation soit dans un état stationnaire.

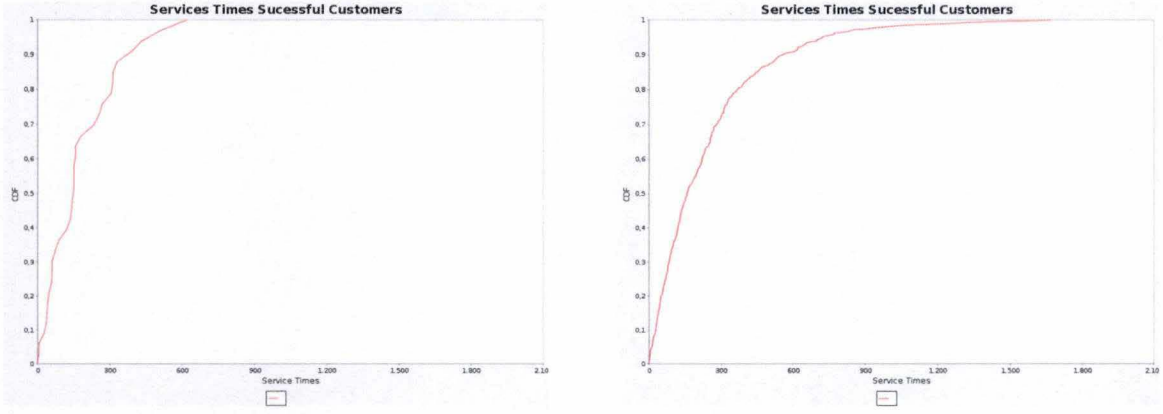



FIGURE 2.4 – Illustration de l'état stationnaire issue du simulateur implémenté : comparaison entre une simulation de 1000 secondes et une simulation de 10000 secondes.

Sur l'illustration 2.4, deux simulations ont été réalisées avec des temps de simulation différents. Le graphique de gauche représente la répartition des temps de séjours des clients avec une durée de simulation de 1000 secondes. Le graphique de droite représente les mêmes temps de séjours mais avec une durée de simulation de 10000 secondes. Nous remarquons que la courbe représentant les temps de séjour sur ces graphiques s'affine lorsque le temps de simulation est plus grand, ceci constitue une indication afin de savoir si nous nous approchons de l'état stationnaire.

### 2.3.3 Analyse des résultats

Nous voyons maintenant la dernière étape de la simulation par événements discrets qui consiste en l'analyse des résultats de la simulation. Pour ce faire, nous avons recours aux outils statistiques dont nous faisons une brève présentation ci-dessous.

**Les statistiques** Les statistiques sont largement employées par la simulation par événements discrets. L'analyse des résultats d'une simulation est généralement effectuée grâce à des outils statistiques. La simulation par événements discrets génère un échantillon d'observations

que nous pouvons donc naturellement manipuler via des outils statistiques. Nous présentons ici les différents outils statistiques les plus fréquemment utilisés par la simulation par événements discrets. L'objectif est de donner au lecteur une idée générale des outils statistiques employés par la simulation par événements discrets.

Les premiers outils utilisés lors de simulations sont des collecteurs d'informations. Ils mémorisent diverses informations telles que le temps de séjour de chaque client ou le nombre total de clients dans le système à un instant $t$. Ces outils fournissent une première vision sur la simulation. Cependant, plus le nombre d'observations sera grand, et plus la lisibilité sera difficile. Nous avons alors recours à d'autres outils issus essentiellement de la statistique descriptive.

La moyenne et l'écart-type sont deux outils statistiques traditionnels couramment employés par la simulation par événements discrets. A la fin d'une simulation, une moyenne des observations permet d'avoir la tendance centrale tandis que l'écart-type peut mesurer la dispersion des ces observations.

Les histogrammes présentent un avantage visuel indéniable, surtout lorsque le nombre d'observations devient élevé. Ils permettent de représenter la distribution des observations, telles que le temps de séjour des clients par exemple. Ainsi nous disposons d'une vue d'ensemble des observations.

Les courbes empiriques sont également utilisées. Les courbes empiriques représentent la probabilité qu'une valeur soit trouvée à une valeur plus petite ou égale à $x$. Ces courbes permettent plus facilement la comparaison entre deux simulations qu'un histogramme. Avec un histogramme, si nous choisissons différentes classes de données, la représentation de l'histogramme varie, ce qui n'est pas le cas avec une courbe empirique. Les courbes empiriques sont monotones et croissantes. La courbe empirique représente la fonction de distribution cumulative (FDC) associée aux valeurs de l'échantillon. Cette fonction avance d'un pas égal à $1/n$ à chaque valeur des $n$ points de l'échantillon. Ci-dessous une définition et un exemple de la courbe empirique.

**Définition d'une courbe empirique 4.**

Étant donné un échantillon $\mathbb{S} = x_1, x_2, ..., x_n$, la courbe empirique de la variable aléatoire $X$ est

$$\hat{F}(x) = \frac{\text{le nombre de } x_i \in \mathbb{S} \text{ pour lequel } x_i \leq x}{n}$$

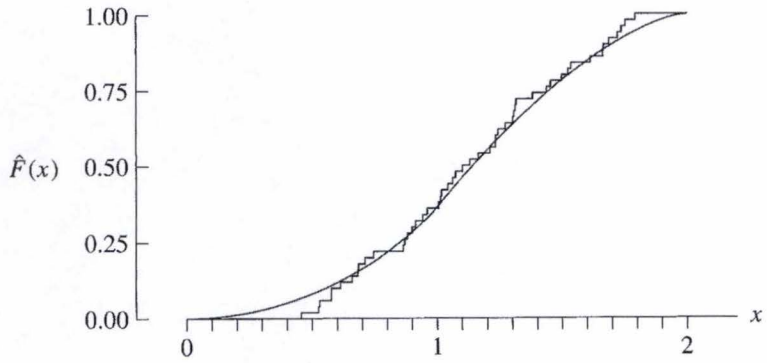



FIGURE 2.5 – Exemple de courbe empirique avec la fonction de distribution cumulative(FDC) théorique correspondante. Figure issue de [3, Chapitre 4, page 170]

Nous voyons sur la figure 2.5 une courbe représentant une fonction de distribution cumulative théorique. Il s'agit de la courbe la plus lisse. L'autre courbe en escalier représente la courbe empirique d'un échantillon essayant de se rapprocher de la courbe théorique. Cet échantillon contient 50 valeurs ce qui est faible comme nombres d'observations (voir 2.3.2). La courbe empirique avance donc d'un pas de 1/50 ce qui donne cette vue en escalier. Si nous augmentons suffisamment le nombre d'observations, la courbe empirique de l'échantillon et la courbe théorique deviendront indiscernables.

# Chapitre 3

# Analyse et Modélisation du système de distribution de contenu vidéo

## Sommaire


**3.1 Méthodologie de la simulation par événements discrets . . . . . 32**
**3.2 Analyse du système de distribution de contenu vidéo . . . . . 34**
3.2.1 Objectifs de l'analyse . . . . . 34
3.2.2 Modèle conceptuel . . . . . 34
3.2.3 Modèle de spécifications . . . . . 35


Dans ce chapitre, nous décrivons une méthodologie pour réaliser des simulations par événements discrets. Nous ferons référence aux différentes étapes de cette méthodologie dans la suite de ce document. Les trois premières étapes de cette méthodologie à savoir les différents objectifs de l'analyse du système de distribution de contenu vidéo, le modèle conceptuel élaboré ainsi que le modèle de spécifications seront présentées dans ce chapitre.

## 3.1 Méthodologie de la simulation par événements discrets

Nous reprenons dans cette section une méthodologie de développement de modèle de simulation. Nous nous en référons à [3, Chapitre 1, page 3]. Le processus de développement d'un modèle de simulation par événements discrets ne peut pas se réduire à une simple succession d'étapes. Cependant un cheminement général existe avec différentes étapes. Nous présentons ce cheminement ci-dessous.

**1) Déterminer les objectifs de l'analyse** Une fois le système qui nous intéresse identifié, nous devons formuler les objectifs de manière claire. En général, quelques phrases suffisent.

**2) Construire le modèle conceptuel** La réalisation du modèle conceptuel du système s'occupe de définir quelles sont les variables utiles, quelles sont les variables liées entre elles, quelle est leur importance. Ensuite ces variables doivent être modélisées de la manière la plus compréhensible possible.

**3) Traduire le modèle conceptuel en un modèle de spécifications** Cette phase consiste à collecter des données sur le système et à les analyser de manière statistique afin de fournir des modèles d'entrée à la simulation. Cette partie inclut aussi le développement d'équations et d'algorithmes nécessaire à la simulation du système.

**4) Traduire le modèle de spécifications en un modèle informatique** Durant cette phase, le choix de langage de programmation doit être fait ainsi que les choix architecturaux de la solution technique. Le choix peut se porter sur un langage de programmation général tel que Java ou sur un langage plus spécialisé en simulation.

**5) Vérifier le modèle** Cette phase consiste à vérifier que le modèle informatique est cohérent avec le modèle de spécifications. Lors de cette phase, un programme informatique existe normalement déjà et il s'agit de vérifier que ce programme simule correctement le système étudié.

**6) Valider le modèle** Cette phase consiste à valider la cohérence entre le modèle informatique et le système analysé. Est-ce que le modèle conceptuel est correct ? Différentes approches existent notamment celles basées sur des statistiques ou celle basée sur la méthode Turing. Cette dernière méthode consiste à présenter à un expert du système étudié des données réelles et des données issues du simulateur. Si l'expert ne sait pas distinguer les données réelles des données de simulation, le simulateur est considéré comme valide.

**7) Concevoir des scénarios de simulation** Cette phase consiste à créer des scénarios de simulation. La tâche n'est pas évidente surtout si le nombre de variables est important. Nous pouvons rapidement avoir une explosion combinatoire. Il faut alors sélectionner les scénarios les plus intéressants.

**8) Réaliser les simulations** Cette phase consiste à effectuer les différentes simulations prévues par les scénarios définis à la phase précédente.

**9) Analyser le résultat des simulations** Cette phase consiste en l'analyse des résultats des simulations. Vu que nous travaillons avec des données qui possèdent un caractère aléatoire, les analyses utilisent la statistique. Nous avons recours aux outils traditionnels de statistiques tels que la moyenne, la variance et les graphiques.

**10) Prendre des décisions** Suivant l'analyse des résultats, des décisions peuvent être prises. Ces décisions concernent par exemple la correction du modèle, l'ajout de variables, ....

**11) Documenter les résultats** Cette phase consiste à documenter et à résumer l'entièreté du travail accompli afin surtout de faciliter des travaux ultérieurs.

## 3.2 Analyse du système de distribution de contenu vidéo

Nous reprenons ici les différentes phases de la méthodologie présentée ci-dessus (voir 3.1) . Les étapes 1 à 3 de cette méthodologie ont été réalisées dans cette étude [5]. Nous reprenons et développons ici ces trois phases qui sont les objectifs de l'analyse, la création du modèle conceptuel et le modèle de spécifications.

### 3.2.1 Objectifs de l'analyse

L'objectif de l'étude est l'analyse du temps de séjour (temps de téléchargement) d'un utilisateur accédant à un service d'enregistrement vidéo en ligne. Le système se compose d'un serveur (ou d'une ferme de serveurs) et de plusieurs utilisateurs. La bande passante à laquelle un utilisateur peut télécharger dépend du nombre d'utilisateurs qui téléchargent simultanément. Deux cas sont traités, l'un où l'utilisateur est limité par sa propre bande passante, l'autre où la bande passante du serveur constitue la limitation. En particulier l'étude s'intéresse à l'influence de l'impatience sur l'utilisateur. Cette impatience est dépendante directement de la vitesse de téléchargement de l'utilisateur et de son temps d'attente.

### 3.2.2 Modèle conceptuel

Nous décrivons dans cette section le modèle conceptuel du système étudié, c'est-à-dire les différentes variables prises en compte pour ce système ainsi que les relations entre ces différentes variables. Il s'agit d'hypothèses que nous prenons. La figure 3.1 illustre via un exemple le modèle conceptuel élaboré pour le système de distribution vidéo. Les différents paramètres et leurs significations sont expliqués ci-dessous.

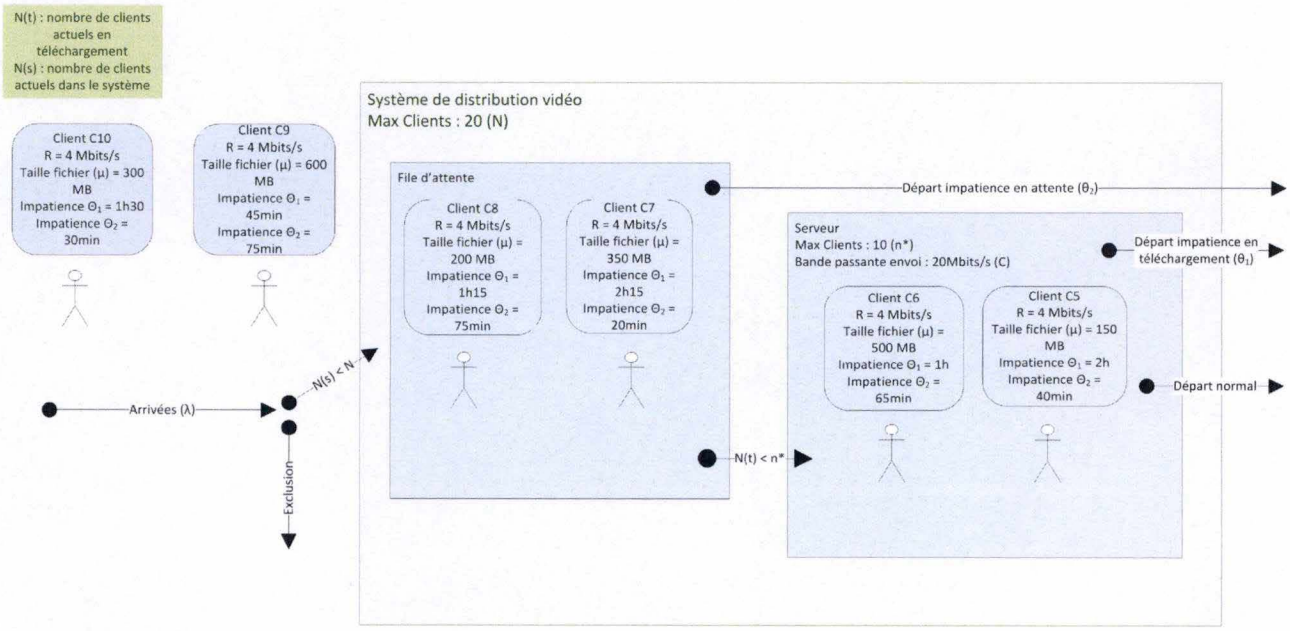



FIGURE 3.1 – Modèle conceptuel du système de distribution vidéo

Le système possède une capacité maximum de $N$ clients aussi appelé $Nmax$ dans la suite de ce document. Au-delà de cette limite les nouveaux clients qui arrivent sont ignorés et rejetés. L'arrivée des clients s'effectue selon un processus de poisson de paramètre $\lambda$ ($\lambda > 0$). Lorsqu'un client arrive, si le système peut encore accueillir un client en téléchargement, le téléchargement du client commence instantanément. Le client est alors mis en service.

Le serveur possède une bande passante pour l'envoi des fichiers vidéos d'une capacité maximum $C$. Cette bande passante est partagée entre les différents clients qui téléchargent. Un nombre maximum de clients qui téléchargent simultanément est fixé à $n*$ aussi appelé $nstar$ dans la suite de ce document. Si le nombre de clients dépasse ce nombre, les clients sont placés dans une file d'attente. Ils attendent qu'une place en téléchargement se libère. Nous appelons ces clients, les clients en attente par opposition aux clients qui téléchargent. Le nombre total de clients en attente et qui téléchargent est donc fini avec un maximum égal à $N$.

Lors du téléchargement d'un fichier, la vitesse de téléchargement pour le client peut être limitée soit par le client soit par le serveur. La capacité maximum de la bande passante de téléchargement d'un client est représentée par $R$. Si la bande passante de téléchargement du client est le facteur limitant, le client télécharge à cette vitesse. Si par contre la limitation de la bande passante provient du serveur, les clients qui téléchargent se partagent la bande passante d'envoi du serveur de manière équitable. Dans ce modèle, la bande passante de téléchargement est identique pour tous les clients. Par exemple, imaginons que nous ayons deux clients avec une bande passante de téléchargement de 4 Mbits/s et un serveur avec une bande passante d'envoi de 20 Mbits/s. Dans ce cas, les deux client peuvent télécharger à 4 Mbits/s, le serveur disposant de 20 Mbits/s peut octroyer sans problème 8 Mbits/s pour ces deux clients. Les clients sont dans ce cas le facteur limitant, ils ne peuvent pas télécharger plus rapidement qu'à du 4 Mbits/s. Si par contre nous avons dix clients avec une bande passante de téléchargement de 4 Mbits/s et un serveur avec un bande passante d'envoi de 20 Mbits/s, le serveur ne peut octroyer que 2 Mbits/s à chaque client. C'est le serveur qui constitue la limitation.

Le taux moyen avec lequel un client termine son téléchargement dépend de la taille du fichier. La taille du fichier est distribuée aléatoirement selon une loi exponentielle de paramètre $\mu$.

Dans le système, un client peut devenir impatient et décider de quitter ce système après un temps aléatoire. L'impatience du client varie en fonction de sa vitesse de téléchargement. C'est pourquoi lorsque le nombre de clients dans le système est plus petit que $n*$, la durée d'impatience est distribuée aléatoirement selon une loi exponentielle de paramètre $\theta_1^{-1}$. Dans les autres cas, il y a plus de $n*$ clients dans le système, l'impatience pour les clients qui téléchargent est identique, par contre la durée d'impatience pour les clients en attente est distribuée aléatoirement selon une loi exponentielle de paramètre $\theta_2^{-1}$. L'impatience de $\theta_1^{-1}$ est plus petite que $\theta_2^{-1}$.

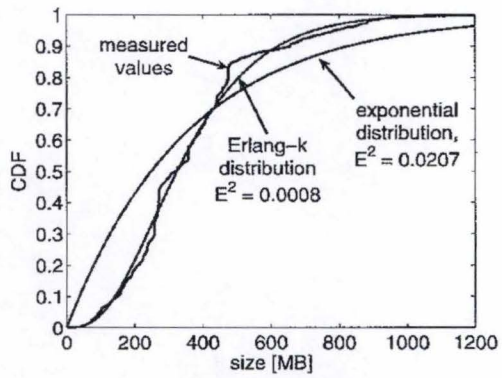
### 3.2.3 Modèle de spécifications

L'objectif de cette étape est de construire des modèles d'entrée de données pour la simulation. Dans le cas qui nous intéresse, nous devons avoir une idée de la bande passante du

serveur, de la bande passante des clients et de la taille des fichiers téléchargés sur le service vidéo afin de coller le mieux possible à la réalité.

**Données d'entrée** Une étude de la taille des fichiers présents sur le service vidéo a été réalisée dans cet article [5]. Il en ressort que la majorité des fichiers sont discrétisés en fichier de 5 minutes. Nous remarquons dans cette étude quatre catégories de programmes télévisés. Beaucoup sont des programmes TV courts d'environ 30 minutes, il s'agit de programmes d'actualités le plus souvent. Cette catégorie est la plus importante. Nous pouvons retrouver les séries télévisées qui occupent la tranche 45-60 minutes. Une dernière catégorie concerne les films et les événements sportifs où la durée est de 90-120 minutes. Cette tranche est plus faible.

Nous nous intéressons à la distribution des fichiers plutôt qu'à la durée des vidéos. L'étude a donc conclu que les fichiers sont distribués suivant une moyenne de 368.31 MegaBytes avec un écart-type de 196.82 MegaBytes. Bien qu'une distribution Erlang reflète mieux la réalité, l'étude a fait l'hypothèse d'utiliser une loi exponentielle pour la distribution de la taille des fichiers. En effet le résidu de la moyenne quadratique est de 0.0207 pour l'exponentielle et de 0.0008 pour l'erlang. Une comparaison entre les mesures réalisées sur les tailles de fichiers, une répartition des tailles de fichiers selon une loi exponentielle et une répartition selon une loi erlang est donnée sur la figure 3.2 ci dessous.




(b) CDF of files sizes

FIGURE 3.2 – Résultats de l'étude : comparaison entre la loi exponentielle, erlang et l'échantillon de l'étude. Figure issue de [5]

Cette investigation a été faite sur la taille des fichiers vidéos présents sur un système d'enregistrement en ligne. Celle-ci a été réalisée en avril 2007 et est constituée d'un échantillon de 11563 fichiers sélectionnés aléatoirement à partir de 19 chaînes de télévision différentes. Les vidéos sont encodées avec une résolution de 512 x 384, un débit vidéo de 750 kbps et un débit audio de 128 kbps. Les mesures ne contiennent que des fichiers de qualité standard c'est-à-dire approximativement 80% de fichiers encodés en format DivX et 20% encodés en format *Windows Media Video* (WMV).

# Chapitre 4

# Analyse pour la construction du simulateur : Modèle informatique

## Sommaire


**4.1 Objectifs du simulateur . . . . . 37**
**4.2 Programmation par événements discrets . . . . . 38**
4.2.1 Construction d'un modèle de simulation *Next-Event* . . . . . 39
4.2.2 Algorithme de simulation *Next-Event* . . . . . 39
**4.3 Analyse pour la construction du simulateur . . . . . 39**
4.3.1 Variables d'états . . . . . 39
4.3.2 Découpe en événements . . . . . 41
4.3.3 Algorithmes pour le changement d'état . . . . . 41


Dans ce chapitre, nous présentons l'analyse réalisée afin de construire le simulateur. Nous nous situons dans la méthodologie proposée (voir section 3.1) à l'étape du modèle informatique. L'analyse débute par la présentation des objectifs du simulateur. Nous continuons ensuite par une introduction à la programmation par événements discrets et en particulier à la méthode *Next-Event* proposant une méthode d'analyse. La section suivante contient l'application de l'approche *Next-Event* afin de réaliser l'analyse pour la construction du simulateur. Nous terminons par des diagrammes d'activités expliquant le fonctionnement du simulateur.

## 4.1 Objectifs du simulateur

L'objectif principal du simulateur est de fournir un outil dynamique capable d'analyser le système de distribution de contenu vidéo étudié par le biais de simulations.

Le système de distribution de contenu vidéo, tel que décrit dans la section 1.4 , a été analysé dans l'article [5] suivant différentes variables de performances notamment l'impatience des utilisateurs. Cette étude a été réalisée selon une approche mathématique suivant une hypothèse sur la distribution de la taille des fichiers. Bien que la distribution de la taille des fichiers semblait s'adapter au mieux avec une distribution de type Erlang (voir 3.2), l'étude a fait l'hypothèse d'utiliser une loi Exponentielle pour représenter la taille de fichiers. Le but

du simulateur est d'implémenter le modèle de simulation décrit dans la section 3.2.2 afin de vérifier dans un premier temps les résultats de cette étude. Dans un second temps, l'objectif est de modifier l'hypothèse sur la distribution de la taille des fichiers afin de tirer des conclusions à propos de l'hypothèse choisie par l'étude.

## 4.2 Programmation par événements discrets

La simulation par événements discrets s'implémente selon l'approche *Next-Event*. Dans cette section nous allons décrire cette méthode. Cette section s'inspire du livre [3, chapitre 5].

L'approche *Next-Event* propose une méthode générale et une terminologie pour construire un modèle de simulation. Elle permet de réduire la complexité de développement d'une application de simulation ainsi que de faciliter l'extension du modèle de simulation.

Différents éléments composent l'approche *Next-Event*. Ils sont décrits ci-dessous.

**État du système** L'état du système est une caractérisation complète du système à un instant donné. Il s'agit d'une capture de l'état du système à un instant particulier. L'état du système est représenté par différentes variables d'états qui contiennent chacune une information sur cet état. Par exemple pour une file d'attente simple, une variable d'état peut contenir le nombre de clients présents dans cette file.

**Événement** Un événement est une action qui provoque la modification de l'état du système. Par définition l'état du système ne peut changer que lorsqu'un événement survient. Chaque événement possède un type. Par exemple pour une file d'attente simple, un événement est l'arrivée d'un client. Un autre événement est le départ d'un client. Nous avons donc deux événements de type différent.

Généralement, nous arrêtons la simulation par événements discrets après un certain laps de temps. Une manière courante d'arrêter le simulateur à un instant donné est de planifier un événement de fin. L'exécution de cet événement provoque alors la fin de la simulation.

**L'horloge** La simulation par événements discrets appartient aux modèles dynamiques (voir chapitre 2), le système évolue donc en fonction du temps. Nous devons dès lors garder une trace du temps qui s'écoule. C'est le rôle joué par l'horloge.

**La planification des événements** Lors d'une simulation par événements discrets, il est nécessaire de savoir à l'avance comment nous progressons dans le temps, quel événement doit être exécuté. L'horloge ne peut pas revenir en arrière, aucun événement ne peut survenir dans le passé. Le mécanisme permettant de savoir à l'avance quel événement va se produire s'appelle *Next-Event*. Ce mécanisme est utilisé conjointement avec la planification des événements pour permettre de connaitre le prochain événement à déclencher.

Les événements sont alors exécutés dans l'ordre croissant de leur planification. Le temps ne s'écoule pas de manière continue, l'horloge avance de manière discontinue d'événement en événement. Dès qu'un événement survient, l'horloge s'arrête le temps d'exécuter l'événement. Ainsi chaque événement survient instantanément peu importe la durée de traitement de cet événement.

**La liste des événements** La liste des événements est la structure de données qui représente l'apparition des événements planifiés dans le temps. Ces événements sont en fait les prochains événements à déclencher. Cette liste doit au moins contenir le prochain événement à déclencher. Cette liste est utilisée par la planification des événements et est triée sur l'horaire de passage des événements.

### 4.2.1 Construction d'un modèle de simulation *Next-Event*

Pour construire un modèle de simulation *Next-Event* il faut réaliser ceci :

1. construire un ensemble de variables d'état qui décrivent l'entièreté du système
2. identifier les différents types d'événements
3. construire les algorithmes qui changent l'état du système dès qu'un événement apparait.

### 4.2.2 Algorithme de simulation *Next-Event*

Une simulation par événements discrets suivant l'approche *Next-Event* suit ces différentes étapes :

- Initialisation : l'horloge est initialisée (souvent à zéro). Le prochain événement à déclencher est placé dans la liste ainsi qu'un événement de fin.
- Traiter l'événement courant : Le prochain événement est retiré de la liste d'événement et est traité.
- Planifier les nouveaux événements : L'événement courant a pu créer d'autres événements qu'il faut donc planifier.
- L'horloge avance dans le temps d'événement en événement jusqu'à satisfaire une condition de fin.

## 4.3 Analyse pour la construction du simulateur

Nous reprenons ici les trois points (4.2.1) de l'approche *Next-Event* afin de construire le simulateur.

### 4.3.1 Variables d'états

Les variables d'états permettent de caractériser le système à un instant donné. Elles sont indispensables au bon fonctionnement du simulateur. L'état du système est ici caractérisé par

les variables suivantes :

- l'horloge de simulation
- la liste des événements
- les clients
- la liste des clients en attente
- la liste des clients en service
- les différents collecteurs de statistiques
- les nombre actuels de clients et les bornes du système.

**L'horloge de simulation** L'horloge de simulation est la variable d'état responsable de fournir le temps actuel.

**La liste des événements** Cette variable contient les prochains événements à exécuter classés par ordre chronologique.

**Les clients** Un client possède différentes propriétés. Nous pouvons regrouper ces propriétés en trois catégories : les variables d'entrées, les variables de travail et les variables de sortie.

Les variables d'entrée :
- le temps d'arrivée du client
- la taille du fichier que le client téléchargera
- la bande passante du client
- le numéro du client
- l'impatience du client en téléchargement (theta1)
- l'impatience du client en attente (theta2)

Les variables de travail :
- le temps écoulé depuis l'arrivée du client
- la taille du fichier restant à télécharger
- la vitesse actuelle de téléchargement

Les variables de sortie :
- le temps de séjour du client
- le temps d'attente du client

**La liste des clients en attente** Cette liste contient à tout moment les clients qui ont été placés en attente par le système.

**La liste des clients en service** Cette liste contient à tout moment les clients qui ont été mis en service par le système.

**Les collecteurs de statistiques** Différents collecteurs de statistiques sont nécessaires afin de collecter des observations tout au long de la simulation. Voici la liste de ces collecteurs :

- le temps de séjour des clients ayant terminé leur téléchargement
- le temps d'attente des clients

- le temps de séjour des clients
- le nombre de clients présents à un instant $t$ dans le système
- le nombre de clients exclus du système
- le nombre de clients servis
- le nombre de clients qui abandonnent en téléchargement
- le nombre de clients qui abandonnent en attente
- le nombre total de clients

**Les nombres actuels de clients et les bornes du système** Il est nécessaire pour le simulateur de connaître le nombre actuel de clients en attente et en service. Les bornes maximales de clients dans le système et en téléchargement doivent aussi être connues. Voici la liste complète de ces variables :

- le nombre actuel de clients en attente
- le nombre actuel de clients en service
- le nombre actuel de clients dans le système
- la borne maximale de clients en téléchargement
- la borne maximale de clients dans le système

### 4.3.2 Découpe en événements

Les événement représentent la seule manière possible pour le système de changer d'état. Les différents événements identifiés sont les suivants :

- L'arrivée d'un client : *Arrival*
- Le départ normal d'un client : *Departure*
- Le départ d'un client impatient alors qu'il était occupé à télécharger : *DepartureImpatience1*
- Le départ d'un client impatient alors qu'il était dans la file d'attente : *DepartureImpatience2*
- La capture du nombre de clients présents dans le système à un instant $t$ : *NbCustomersInSystem*
- La fin de la simulation : *EndOfsim*

Nous utiliserons désormais les noms en *italique* pour désigner ces événements.

### 4.3.3 Algorithmes pour le changement d'état

Les changements d'états s'opèrent uniquement lorsqu'un événement survient. Les algorithmes de changements d'états décrivent le comportement de ces événements. Nous avons utilisé la notation UML des diagrammes d'activités pour représenter le traitement effectué par le simulateur lors du déclenchement des différents événements. Nous décrivons ci-dessous la syntaxe et la sémantique de cette notation. Ensuite nous présentons les diagrammes d'activités des différents événements gérés par le simulateur.

**Notation UML des diagrammes d'activités** Les diagrammes d'activités sont un excellent outil pour comprendre le fonctionnement d'un système ou d'un processus. Ils mettent en évidence le flux des activités réalisées par un processus. Voici la syntaxe utilisée dans ce document :

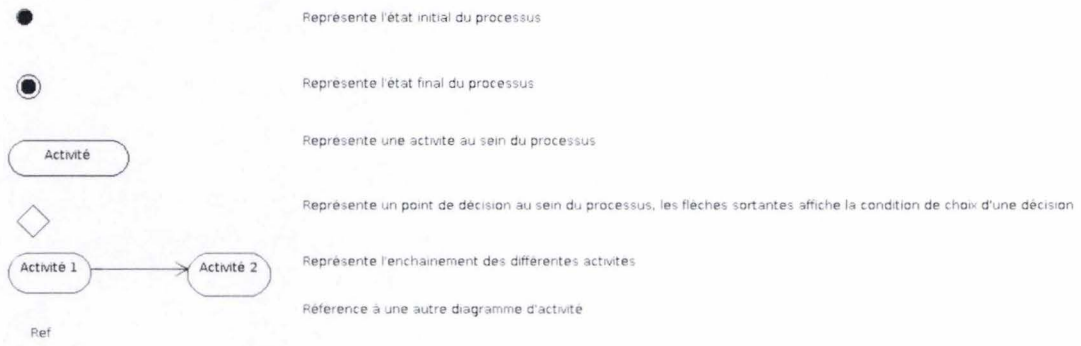



FIGURE 4.1 – Diagrammes d'activités : légende

**Diagramme d'activité : fonctionnement du simulateur**

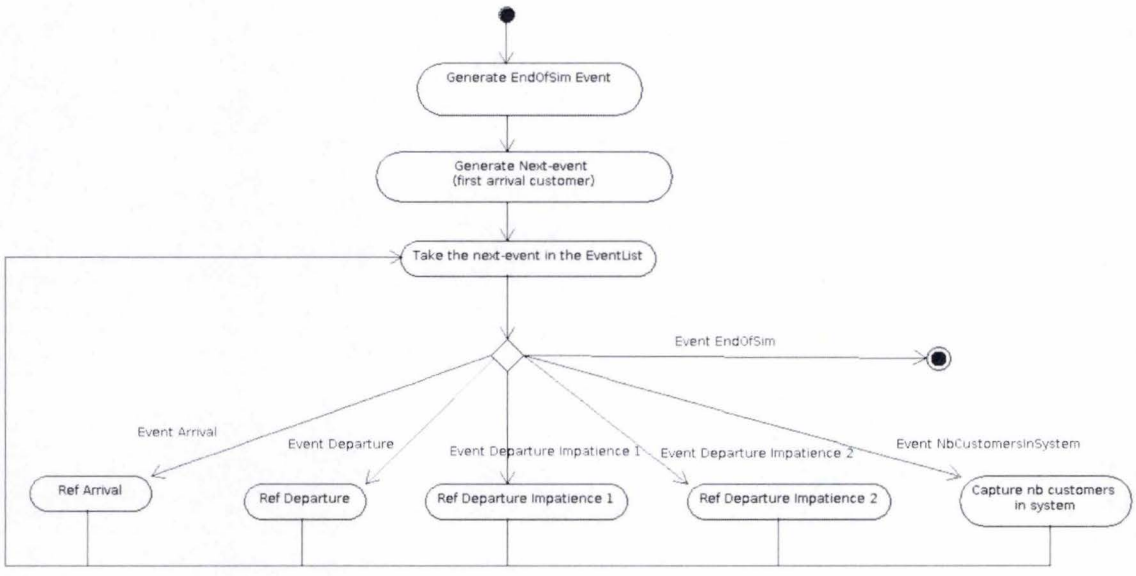



FIGURE 4.2 – Comportement du simulateur : schéma général

La figure 4.2 montre l'enchainement des actions effectuées par le simulateur. Le simulateur utilise l'algorithme *Next-Event* introduit dans la section 4.2.2. Il commence donc par une phase d'initialisation. Celle-ci consiste en la création d'un événement de fin de simulation ainsi que la

création du prochain événement. L'événement de fin permet de s'assurer de la terminaison de la simulation après un laps de temps défini. Le prochain événement est, quant à lui, nécessaire pour amorcer la simulation. Il s'agit d'un événement *Arrival* signifiant l'arrivée du premier client. Les deux premières activités présentes sur la figure 4.2 illustre cette phase d'initialisation. Ensuite le simulateur sélectionne le prochain événement à déclencher et l'exécute. Il existe six types d'événements que le simulateur peut exécuter comme défini dans la section 4.3.2.

### Diagramme d'activité : événement *Arrival*

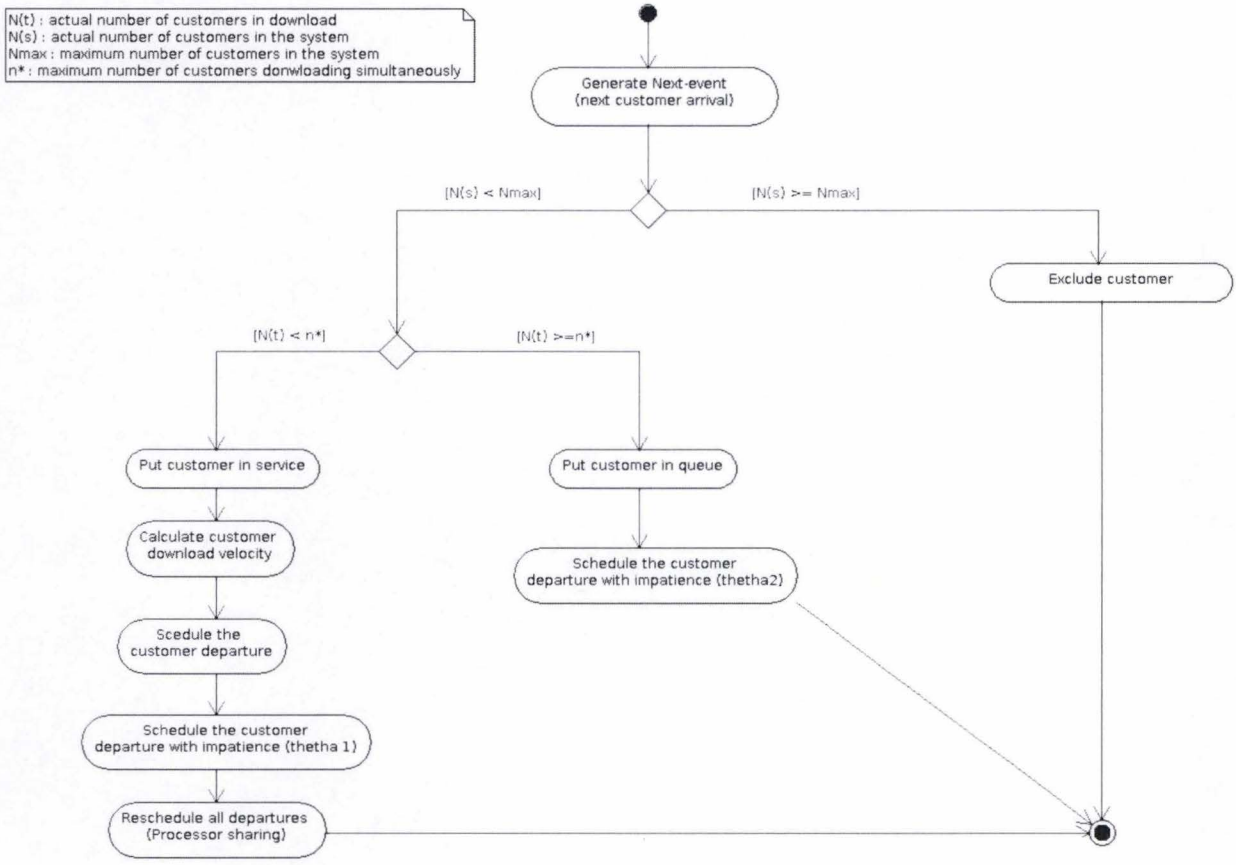



FIGURE 4.3 – Comportement du simulateur : événement *Arrival*

La figure 4.3 décrit le processus de traitement d'un événement *Arrival*. La gestion de cet événement commence par la génération d'un autre événement *Arrival*. Il faut savoir que pour effectuer une simulation, des arrivées de clients sont nécessaires afin d'alimenter le système. Sans arrivées de clients, rien ne se passe à l'intérieur du système et la simulation ne peut pas enregistrer d'observations intéressantes. Dès l'arrivée d'un client, l'arrivée d'un autre client est par conséquent planifiée afin de s'assurer que la simulation puisse continuer. Cette action est la première activité illustrée sur la figure 4.3 et correspond à la planification de nouveaux événements de l'algorithme de simulation *Next-Event* (voir section 4.2.2).

Ensuite le client actuel, celui qui vient d'arriver, est traité. Cette action consiste à placer le client soit en service, soit en attente ou bien de l'exclure. Ce placement est déterminé par le nombre de clients actuels à l'intérieur du système et les bornes maximums de clients en téléchargement ($n*$) et dans le système ($nmax$). Dans le cas d'une mise en attente du client, un événement *DepartureImpatience2* est planifié pour le client. En effet, un client placé en attente quitte le système après un temps d'impatience défini au préalable. Dans le cas d'une mise en service du client, un événement *Departure* ainsi qu'un événement *DepartureImpatience1* sont planifiés. Cela signifie que le client peut partir soit normalement après avoir terminé son téléchargement, ou après un temps d'impatience défini au préalable. Lors de la mise en service d'un client, la vitesse à laquelle le client télécharge son fichier doit être calculée. Vu que le serveur partage sa bande passante entre les différents clients, l'ajout d'un client en service provoque ainsi un nouveau calcul des vitesses de téléchargement des clients et donc de leurs départs. Les événements *Departure* doivent être replanifiés. Ces deux calculs sont expliqués ci-dessous.

**Calcul de la vitesse de téléchargement** Comme énoncé dans la section 3.2.2, la vitesse de téléchargement d'un client peut être limitée soit par la bande passante de téléchargement du client lui-même, soit par la bande passante d'envoi du serveur. Le calcul de la vitesse de téléchargement a donc pour but d'identifier si le goulot d'étranglement se situe au niveau du serveur ou du client. La stratégie adoptée par le serveur est la technique du *Processor Sharing* (voir ci-dessous). La bande passante du serveur est donc divisée entre les différents clients en service de manière égale. Si celle-ci est inférieure à la bande passante du client, le client téléchargera à cette vitesse. Si par contre la bande passante du serveur est supérieure à celle du client, le client téléchargera à sa vitesse.

**Calcul des départs** Le serveur utilise la technique du *Processor Sharing*. Cela signifie que dès qu'un événement lié au service survient (le départ d'un client, l'arrivée d'un client en service, ...), le calcul des départs doit être actualisé. L'actualisation des départs s'effectue de la manière suivante. La liste d'événements est parcourue à la recherche de tous les événements de départs déjà prévus. La vitesse de ceux-ci est alors recalculée suivant le nombre actuel de clients en service. Un nouveau départ est ensuite prévu pour ces clients. Il faut savoir que ces clients ont déjà commencer à télécharger une partie de leur fichier. Le calcul des départs prend donc en compte la taille de fichier restant à télécharger et non la taille du fichier complet.

**Comportement du simulateur : événement *Departure* et *DepartureImpatience1***

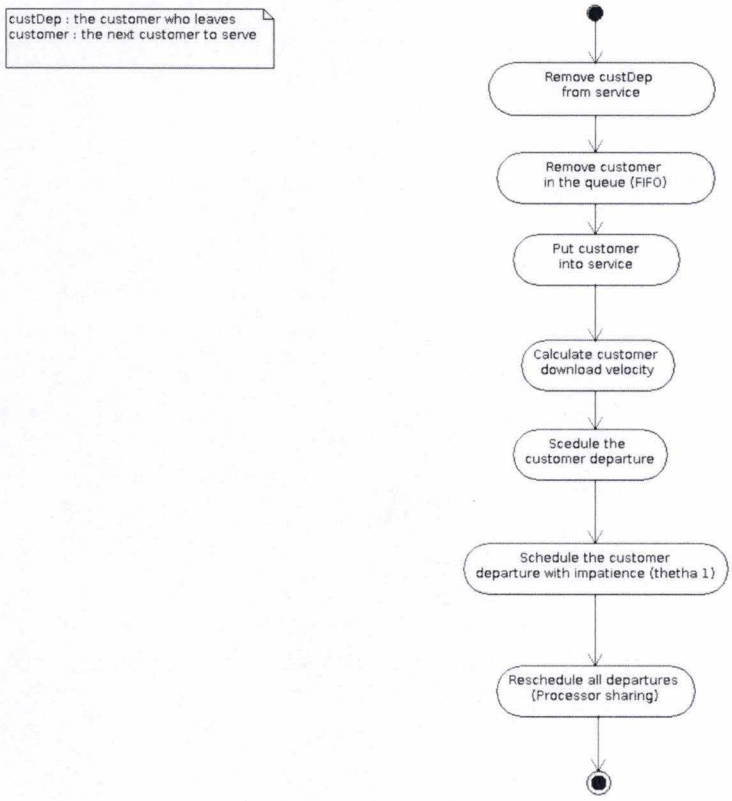



FIGURE 4.4 – Comportement du simulateur : événement départ normal et avec impatience theta1

La gestion d'un événement *Departure* et *DepartureImpatience1* exécute le même traitement. C'est pourquoi un seul graphique représente ces deux types de départ. Nous les avons toutefois séparé lors de la découpe en événements puisqu'il peut s'avérer utile de collecter des observations différentes pour ces deux types d'événements. La figure 4.4 illustre le processus de traitement. Ce processus commence par le retrait du client qui part de la liste des clients en service. Ensuite un client est sélectionné dans la file d'attente pour être mis en service. La discipline utilisée par la file d'attente est *FIFO*. Les clients sont donc sélectionnés selon leur ordre d'arrivée. La vitesse de téléchargement pour ce client est ensuite calculée (voir 4.3.3) . Un événement *Departure* et *DepartureImpatience1* sont alors planifiés pour ce client. Ensuite les départs sont replanifiés ( voir 4.3.3).

**Comportement du simulateur : événement *DepartureImpatience2***

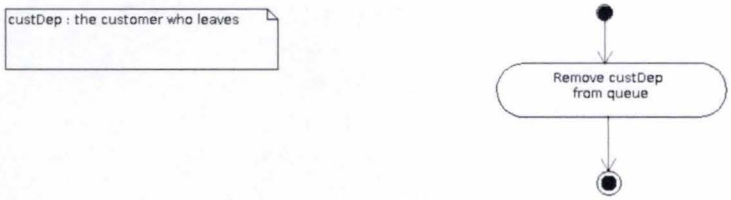



FIGURE 4.5 – Comportement du simulateur : événement *DepartureImpatience2*

La figure 4.5 illustre le traitement effectué lors d'un événement *DepartureImpatience2*. Le traitement consiste simplement à retirer le client de la file d'attente.

# Chapitre 5

# Implémentation du simulateur

## Sommaire


**5.1 SSJ** . . . . . **47**
5.1.1 Description générale de SSJ . . . . . 47
5.1.2 Notation UML : Diagramme de classes . . . . . 48
5.1.3 Description détaillée de SSJ . . . . . 50
**5.2 Diagramme de classes du simulateur** . . . . . **54**
**5.3 Interface graphique** . . . . . **58**
**5.4 Mécanismes de performances et d'efficacité appliqués au simulateur** . . . . . **61**


Dans ce chapitre, nous expliquons le choix du langage de programmation ainsi que les librairies utilisées. Pour le choix du langage nous avons opté pour *Java*. *Java* est un langage de programmation à usage général. Il n'offre donc pas comme les langages de simulation spécialisés des facilités pour développer un logiciel de simulation. Les outils de simulation, de statistiques ne sont pas présents. Cependant le langage *Java* offre de nombreuses librairies sur lesquelles s'appuyer.

Nous avons choisi d'utiliser la librairie *Stochastic Simulation in Java* (SSJ [1]). Nous présentons ci-dessous cette librairie et plus particulièrement les outils que nous avons utilisés.

Ensuite nous présentons et expliquons le diagramme de classes du simulateur. Nous terminons par discuter de l'interface graphique ainsi que de différents moyens mis en œuvre pour rendre le simulateur performant et efficace.

## 5.1 SSJ

### 5.1.1 Description générale de SSJ

Voici une description de la librairie *SSJ* donnée par ses concepteurs :

1. `http://www.iro.umontreal.ca/~simardr/ssj/indexf.html`

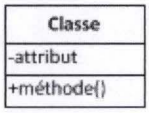
"SSJ est une bibliothèque Java, développée au Département d'Informatique et de Recherche Opérationnelle (DIRO) de l'Université de Montréal, et fournissant des outils pour la simulation stochastique. SSJ contient différents paquetages offrant des outils pour générer des valeurs aléatoires uniformes et non uniformes, pour calculer différentes quantités liées à des lois de probabilité, effectuer des tests d'ajustement ("goodness-of-fit"), appliquer des méthodes de type quasi-Monte Carlo, et programmer des simulations à événements discrets avec vision par événements ou par processus." Cette description est issue du site web [2] de la librairie.

### 5.1.2 Notation UML : Diagramme de classes

Nous présentons ici les notions essentielles de la notation UML des diagrammes de classes. L'objectif n'est pas d'être exhaustif mais de fournir les bases nécessaires pour comprendre les diagrammes de classes réalisés dans les sections suivantes. Nous avons utilisé cette notation dans la suite de ce document pour décrire l'architecture de la librairie *SSJ* ainsi que l'architecture du simulateur. Elle est particulièrement adaptée à la programmation orientée objet qui est utilisée aussi bien par la librairie *SSJ* que par le simulateur. Si vous êtes familier avec cette notation, cette section peut être passée.

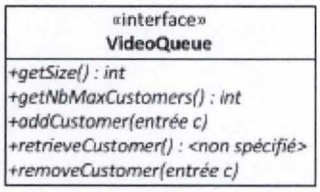
Les diagrammes de classes permettent de représenter les structures statiques d'une application informatique ainsi que les relations entre celles-ci. Il faut savoir qu'une application informatique est découpée en différentes structures, chacune responsable de la gestion d'un concept. Ci dessous une légende de la syntaxe et sémantique de la notation UML des diagrammes de classes utilisée dans ce document.

#### Syntaxe et sémantique des diagrammes de classes

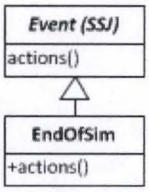



Une classe représente un concept, une structure de donnée utilisée par l'application informatique. Chaque classe est en fait un type de données, rassemblant des propriétés communes à un concept ainsi que des opérations communes. Les propriétés sont appelées attributs tandis que les opérations sont appelées méthodes. Les attributs et les méthodes possèdent une visibilité. La visibilité détermine l'accessibilité à un attribut ou à une méthode. Le signe "-" indique que l'attribut ou la méthode ne sera accessible qu'à l'intérieur de la classe. Le signe "#" indique que l'attribut ou la méthode sera accessible qu'à l'intérieur de la classe et à ses enfants(voir héritage ci-dessous). Le signe "+" indique que l'attribut ou la méthode sera accessible de manière publique. Nous pouvons créer des objets à partir d'une classe. Une classe étant en fait une fabrique d'objet. On parle dans la programmation orientée objet d'instanciation lors de la création d'un objet à partir d'une classe. Une classe peut avoir son nom en *italique*, il s'agit alors d'une classe abstraite. Ce type de classe ne peut pas être instancié. Elle ne fixe que certaines règles que devront respecter ses enfants (voir héritage ci-dessous).

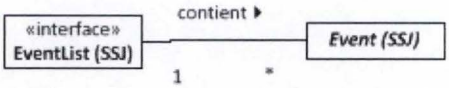
2. `http://www.iro.umontreal.ca/~simardr/ssj/indexf.html`



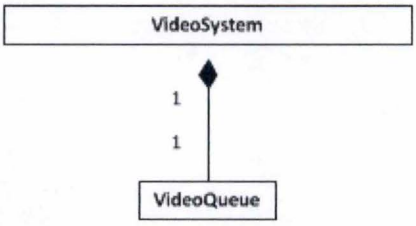

Une interface est composée d'un ensemble de méthodes non implémentées. Les classes peuvent alors s'engager à fournir une implémentation, un traitement pour ces méthodes. D'autres classes peuvent exiger l'emploi de cette interface pour leur utilisation. Ceci permet de fixer un contrat à respecter entre l'utilisation de méthodes et leurs implémentations. Les méthodes d'une interface ont toujours une visibilité publique. Si une classe implémente une interface, cela signifie que cette classe fournit une implémentation de toutes les méthodes définies par l'interface. L'avantage des interfaces est que plusieurs classes peuvent alors définir une implémentation de celle-ci. Les classes appelantes utilisent l'interface. Nous pouvons donc changer d'implémentation sans devoir modifier les classes appelantes.




Un concept important dans la programmation orientée objet est la relation d'héritage. Ceci permet de créer une organisation de classe dans le but de faciliter la réutilisation du code. La classe située au-dessus de la relation constitue le parent, la classe située en-dessous de la relation d'héritage constitue l'enfant. Une classe enfant spécialise une classe parent. Ceci veut dire qu'une classe enfant récupère automatiquement les attributs et les méthodes de son parent. Il peut ensuite les compléter par ses propres attributs et méthodes. Si la relation d'héritage est en pointillé, cela signifie que la classe parent est une interface. La classe enfant fournit alors les méthodes exigées par la classe parent (voir interface ci-dessus).




Les classes sont reliées entre elles par des relations. Un type de relation fréquent dans la notation des diagrammes de classes UML est la relation binaire. Cette relation est accompagnée d'un nom, d'un sens de lecture et d'une multiplicité à chaque extrémité. Ces relations permettent de préciser les liens entre les différentes classes. Ceci se traduit dans l'application informatique par des références autrement dit un accès entre ces classes. Ceci se lit de la manière suivante : un événement est contenu dans une seule liste d'événements. Une liste d'événements contient plusieurs événements.



Ceci représente une relation de composition. La classe *VideoSystem* représente le composant tandis que la classe *VideoQueue* représente le composé. Il s'agit d'une relation binaire spécialisée. Dans ce type de relation, les deux éléments existent ensemble ou n'existent pas.

### 5.1.3 Description détaillée de SSJ

La librairie *SSJ* propose de nombreux outils pour réaliser des simulations par événements discrets. Nous détaillons ci-dessous les outils de cette librairie que nous avons utilisés.

**Générateur de nombres aléatoires** Nous savons que la simulation par événements discrets s'appuie sur des générateurs de nombres aléatoires (voir 2.3.1). La librairie *SSJ* propose donc plusieurs générateurs de nombres aléatoires. Le générateur de nombres aléatoires que nous avons choisi s'appelle *MRG32k3a*. Il a été utilisé pour générer les temps d'arrivées, les temps de services et les temps d'impatience. Ce générateur est basé sur l'algorithme de *Lehmer* (voir section 2.3.1).

Il possède plusieurs *streams*, ce qui signifie que nous pouvons utiliser une seule instance de ce générateur pour des fins différentes. Les *streams* sont une facilité de certains générateurs de nombres aléatoires permettant avec un seul générateur de nombres aléatoires d'avoir plusieurs séquences de nombres aléatoires disjointes. Ainsi un seul générateur de nombres aléatoires peut s'utiliser dans des contextes différents tels que le temps d'arrivée et le temps de service par exemple.

Il possède une période de longueur $2^{191}$ ce qui est confortable (voir 2.3.1) . Ce générateur a été choisi car il a été fortement testé et recommandé par la communauté de la librairie *SSJ*.

**Fonction de distribution inverse** Nous avons vu dans la section 2.3.1 que pour générer une variable aléatoire liée à une loi de probabilité, nous devons employer la fonction de distribution inverse. La classe *ExponentialGen* de la librairie *SSJ* propose une implémentation de la fonction inverse appliquée à une loi exponentielle.

La figure 5.1 montre un diagramme de classe partiel donnant la structure de la librairie *SSJ* en ce qui concerne la génération des nombres aléatoires et la fonction de distribution inverse. Nous utilisons la notation UML des diagrammes de classes introduite dans la section 5.1.2.

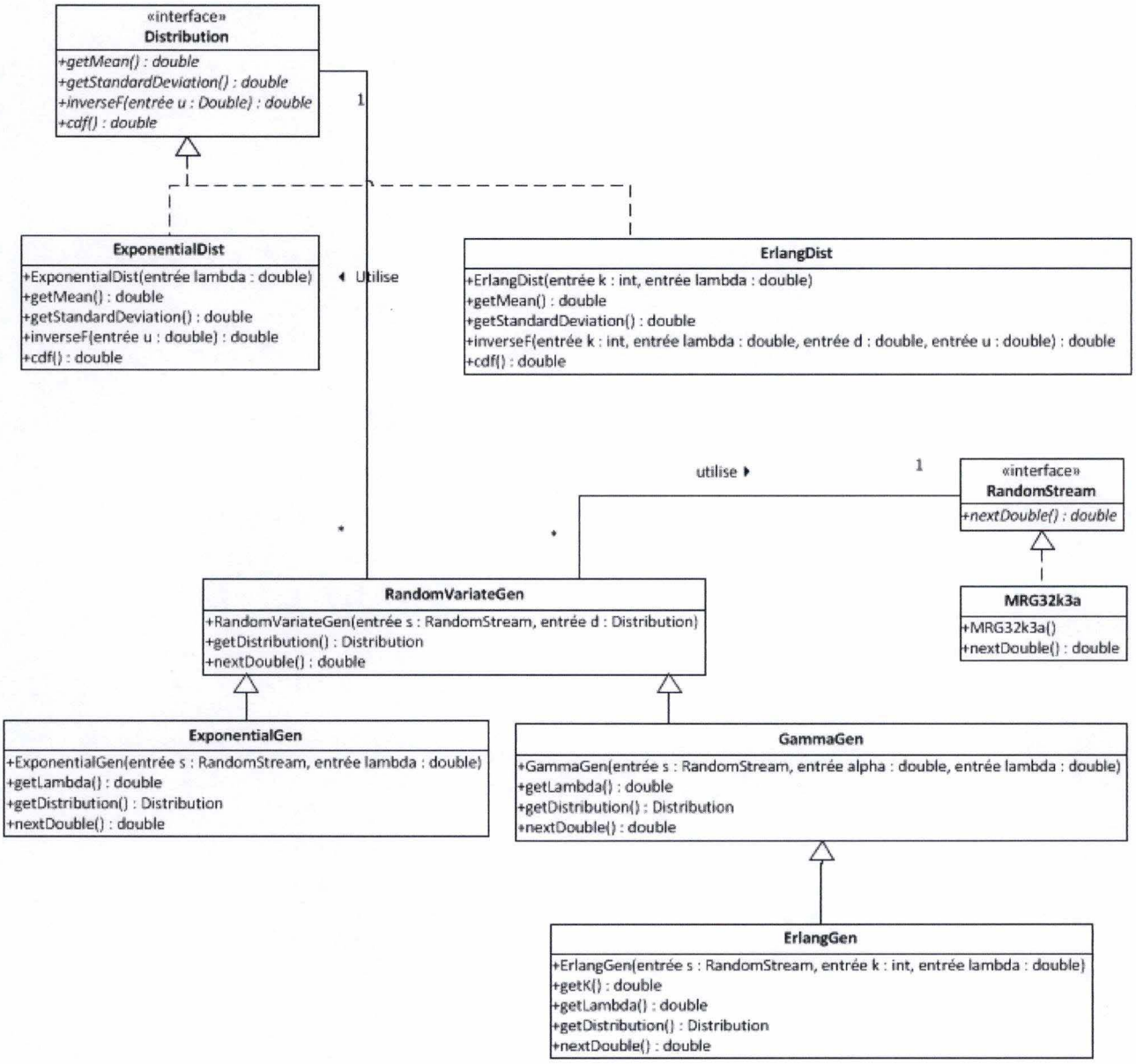



FIGURE 5.1 – Diagramme de classe de la librairie SSJ (partie Générateur de nombres aléatoires)

La figure 5.1 illustre les différentes classes créées par la librairie *SSJ* afin de générer des variables aléatoires pour la simulation par événements discrets. Nous remarquons que pour la gestion des lois de probabilités, la librairie propose une interface *Distribution*. Différentes implémentations pour cette interface sont disponibles notamment pour l'Exponentielle ou l'Erlang. De manière identique, nous observons une interface *RandomStream* possédant une méthode *nextDouble()*. C'est cette méthode qui est en charge de renvoyer un nombre aléatoire compris entre 0.0 et 1.0. Les générateurs de nombres aléatoires notamment le générateur *MRG32k3a* que nous avons utilisé fournissent une implémentation pour cette interface. La classe *RandomVariateGen* permet le regroupement entre la génération d'un nombre aléatoire et l'association de cette génération avec une loi de probabilité. L'appel à la méthode *nextDouble()* de cette classe provoque la génération d'un nombre aléatoire, celui-ci est ensuite passé à la fonction de distribution inverse de la loi de probabilité définie. La classe *ExponentialGen* ou *ErlangGen* sont des classes spécialisées pour la loi exponentielle et la loi Erlang. Elles disposent donc de méthodes supplémentaires pour l'Exponentielle et pour l'Erlang.

**L'horloge, la liste d'événements et les événements** La figure 5.2 présente un diagramme partiel de classe illustrant la structure et les relations entre les classes de la librairie *SSJ* pour la partie concernant la gestion des événements.

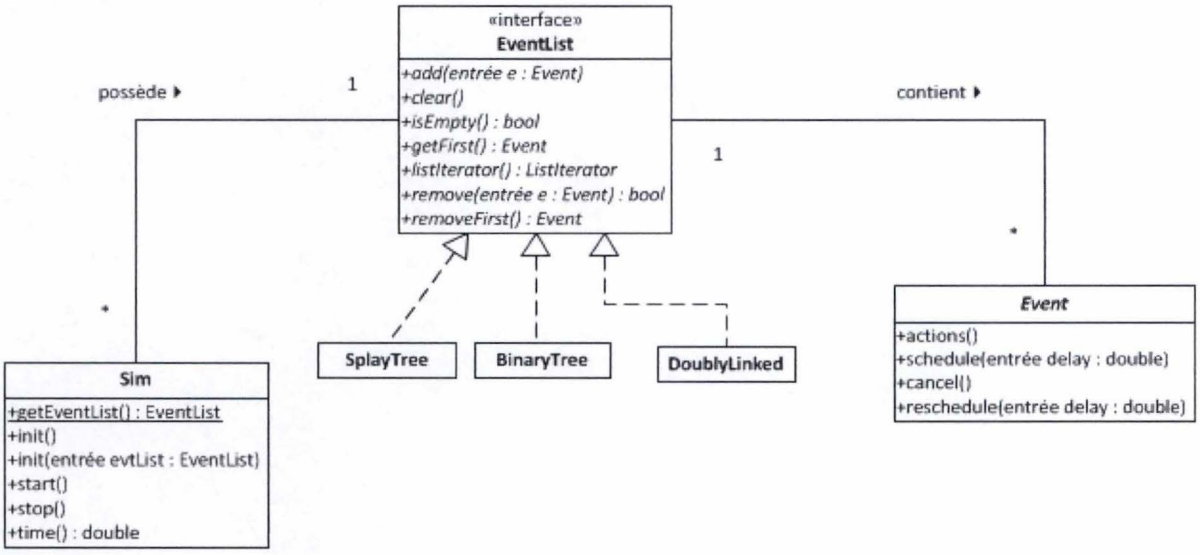



FIGURE 5.2 – Diagramme de classe de la librairie SSJ (partie Événement)

Sur la figure 5.2, nous pouvons distinguer une classe *Sim* qui implémente l'horloge de simulation. Elle possède un lien vers la liste des événements. *SSJ* offre une interface *EventList* définissant une liste d'événements. SSJ propose pour la gestion de la liste d'événements plusieurs implémentations telles que la liste chaînée, l'arbre binaire ou encore l'arbre *splay*. Nous avons utilisé cette dernière implémentation. Elle est similaire à un arbre binaire de recherche excepté le fait que chaque opération effectue au préalable une action *splay*. Cette action consiste à remonter à la racine de l'arbre la valeur désirée. Cette action *splay* effectue un rebalancement de l'arbre. Cependant celle-ci est moins coûteuse que pour un arbre rouge/noir et permet

d'éviter le cas où un parcours de l'arbre en entier est nécessaire. Pour plus de détails sur cette implémentation, veuillez vous référer à [1]. Les opérations, suivant la notation de complexité en temps $O$, sont effectuées en $O(log(n))$. Nous avons utilisé cette implémentation car elle offre l'avantage d'effectuer les opérations de recherche en $O(log(n))$ contre du $O(n)$ pour les autres implémentations. Il faut savoir, que le simulateur pour gérer le *Processor Sharing*, doit parcourir la liste des événements à la recherche des événements de départs afin de replanifier ceux-ci. Lors du départ d'un client, il est aussi nécessaire de retirer de la liste des événements tous les événements créés pour ce client. Une implémentation efficace au niveau des opérations de recherche s'avère donc utile.

Pour réaliser une simulation par événements discrets, nous devons définir des événements ainsi que leurs comportements. *SSJ* propose une classe abstraite *Event* disposant d'une méthode abstraite *actions*. L'implémentation de cette méthode permet de définir le comportement d'un événement c'est-à-dire le traitement qui sera effectué lors du déclenchement de l'événement. Pour créer nos événements, nous n'avons donc qu'à créer des classes héritant de celle-ci et implémenter la méthode *actions* qui réalise l'exécution de l'événement.

**Statistiques** Durant la simulation, nous devons collecter des observations afin de pouvoir réaliser des statistiques par la suite. Voici un diagramme de classe partiel de la librairie *SSJ* en ce qui concerne la collecte des statistiques et la construction de graphiques pour l'analyse des résultats.

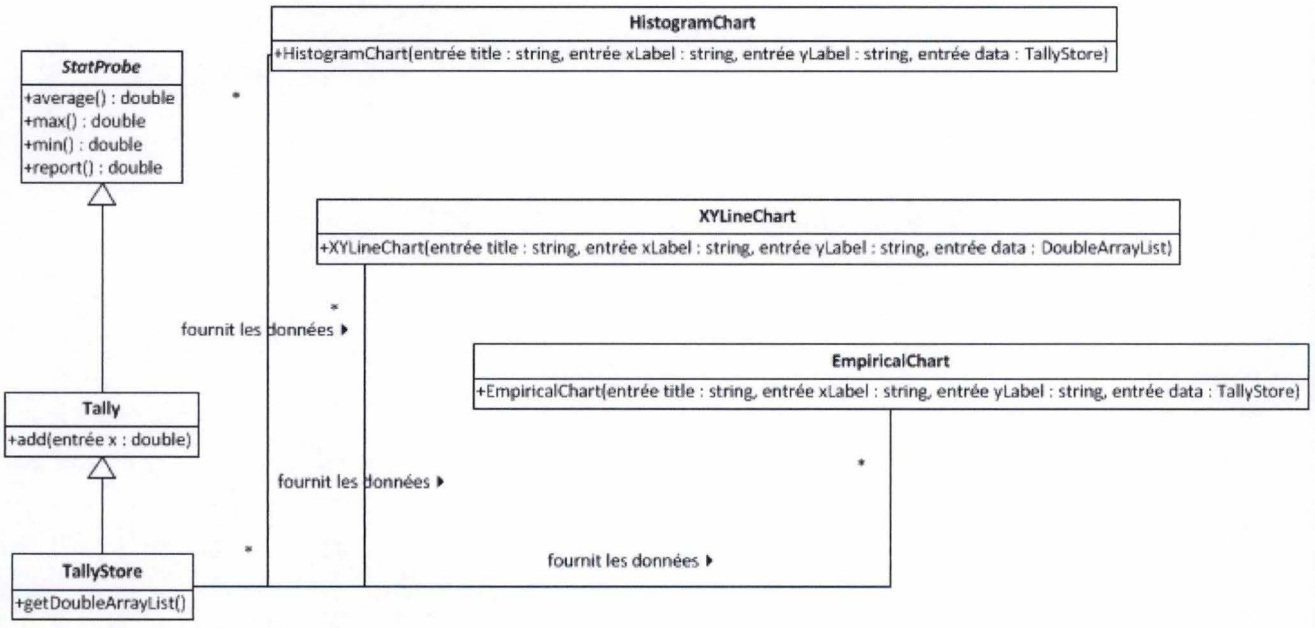



FIGURE 5.3 – Diagramme de classe de la librairie SSJ (partie Collecte de statistiques)

La classe *StatProbe* de la librairie *SSJ* est une classe abstraite d'où découlent les collecteurs d'observations implémentés par *SSJ*. La classe *Tally* est l'une de ces classes. Cette classe possède une méthode d'ajout d'observation ainsi que différentes méthodes implémentant les outils statistiques traditionnels tel que la moyenne et l'écart-type. La classe *Tally* ne mémorise pas les observations collectées. La classe *TallyStore* est identique à la classe *Tally* mais elle

permet de récupérer les observations collectées sous forme d'une liste doublement chaînée. Contrairement à la classe *Tally*, elle mémorise les observations.

Nous l'avons vu, les résultats des simulations par événement discrets sont traités par des outils statistiques et sont représentés par graphiques. La librairie *SSJ* s'est appuyée sur la librairie *JFreeChart*[3] pour développer ses outils graphiques. La librairie *SSJ* propose plusieurs types de graphiques. Nous en avons retenu les suivants. La classe *EmpiricalChart* permet la création de courbes empiriques telles que définies dans la section 2.3.3. La classe *HistogramChart* permet de créer des histogrammes. La classe *XYLineChart* permet la création d'un graphique sur base de coordonnées $X$ et $Y$.

## 5.2 Diagramme de classes du simulateur

Dans cette section, nous présentons le diagramme de classe du simulateur ainsi qu'une justification de cette découpe.

Le simulateur s'est découpé naturellement en différents composants présents dans la simulation par événements discrets. Regardons plus en détail chacune des classes du diagramme présent sur la figure 5.4.

Nous retrouvons les classes *Sim*, *EventList*, *SplayTree*, *Event* et *RandomVariateGen* de la librairie SSJ. Nous les avons dessinées afin de montrer les différentes relations avec les classes créées pour réaliser le simulateur. Ces classes ont été décrites et expliquées ci-dessus.

La classe *VideoEvent* est une classe abstraite possédant des références, c'est-à-dire des points d'entrée vers les différents éléments constituant le simulateur. Ces références donnent accès au système (*VideoSystem*), à la file d'attente (*VideoQueue*), au serveur (*VideoServer*) et aux collecteurs de statistiques (*StatsCollectors*). Lors de son exécution, un événement a besoin d'interagir avec les différents composants du système tels que le serveur (*ViderServer*) ou les collecteurs de statistiques (*StatCollectors*). Il s'agit par exemple de demander au serveur si une place en service est disponible ou de transmettre au collecteur de statistiques une observation. L'intérêt de cette classe est de proposer à ses classes enfants un canal de communication vers tous les composants du simulateur. Nous pouvons dès lors créer facilement un nouveau type d'événement ayant accès aux différents composants du simulateur.

La classe *Arrival* traite l'arrivée d'un client tel que décrit par le diagramme d'activité 4.3. La méthode *actions()* réalise ce traitement à savoir la planification d'un nouvel événement *Arrival*, le placement du client courant, le calcul de la vitesse de téléchargement du client et le calcul de son départ. Pour plus de détails sur ce traitement, référez-vous à la section 4.3.3. C'est également pendant ce traitement que sont mis à jour différents compteurs statistiques que sont le nombre de clients exclus et le nombre total de clients.

La classe *Departure* traite le départ normal d'un client tel que décrit par le diagramme d'activité 4.4. La méthode *actions()* réalise ce traitement à savoir le retrait de ce client du service, la sélection d'un nouveau client dans la file d'attente suivant la discipline FIFO et la planification du départ de ce client. Pour plus de détails sur ce traitement, référez-vous à la section

3. `http://www.jfree.org/jfreechart`

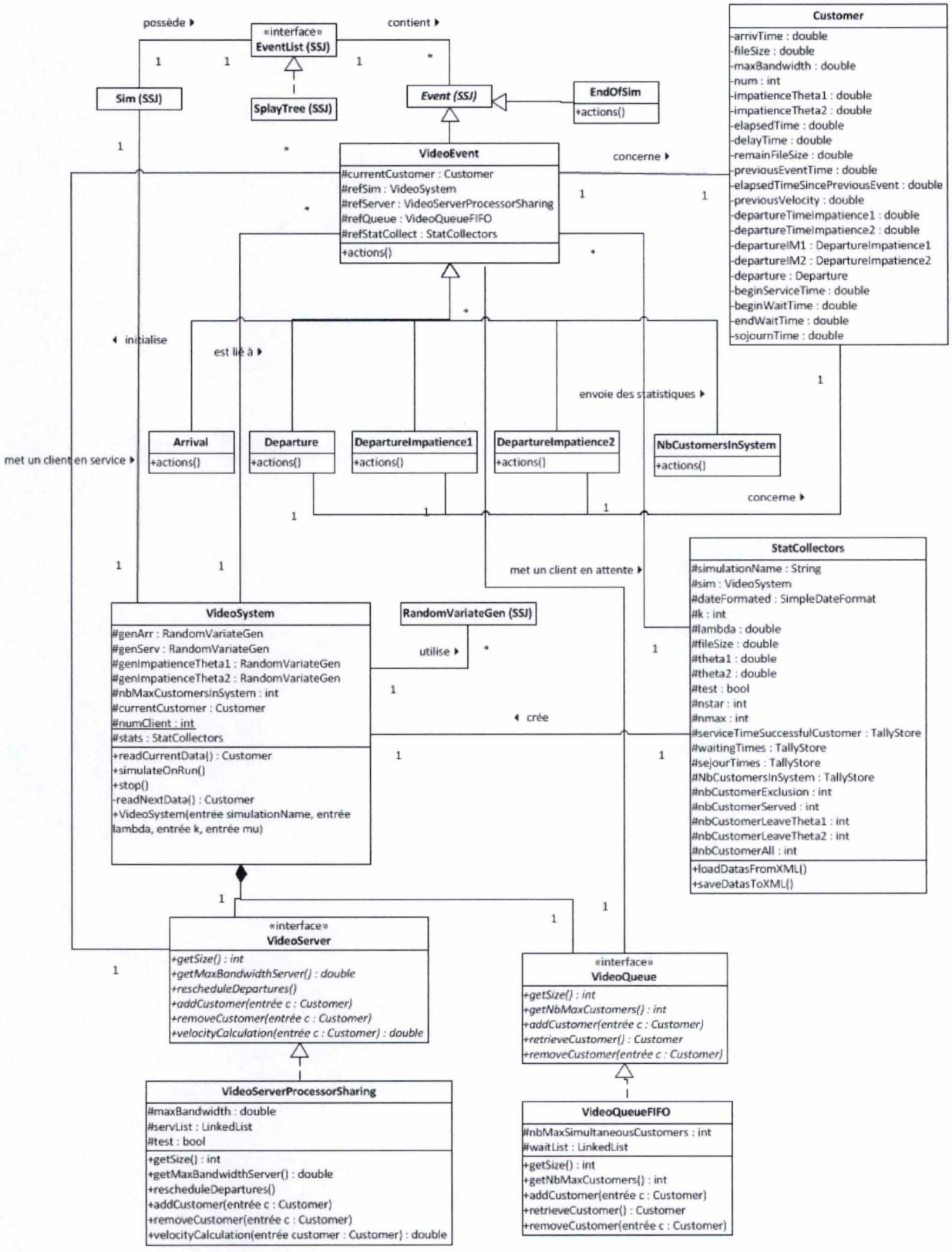



FIGURE 5.4 – Diagramme de classe du simulateur

4.3.3. C'est également pendant ce traitement que sont mis à jour différents collecteurs statistiques et compteurs d'observations. Ainsi le temps de séjour du client, son temps de service et son temps d'attente sont mémorisés. Le compteur du nombre de clients servis est mis à jour.

La classe *DepartureImpatience1* traite le départ d'un client suite à de l'impatience alors qu'il télécharge un fichier. Le traitement est réalisé tel que décrit par le diagramme d'activité 4.4. La méthode *actions()* réalise ce traitement à savoir le retrait de ce client du service, la sélection d'un nouveau client dans la file d'attente suivant la discipline FIFO et la planification du départ de ce client. Pour plus de détails sur ce traitement, référez-vous à la section 4.3.3. C'est également pendant ce traitement que sont mis à jour différents collecteurs statistiques et compteurs d'observations. Ainsi le temps de séjour du client, son temps de service et son temps d'attente sont mémorisés. Le compteur du nombre de clients qui abandonnent pendant le téléchargement est mis à jour.

La classe *DepartureImpatience2* traite le départ d'un client suite à de l'impatience alors que celui-ci se trouve dans la file d'attente. Le traitement est réalisé tel que décrit par le diagramme d'activité 4.5. La méthode *actions()* réalise ce traitement à savoir le retrait de ce client de la file d'attente. C'est également pendant ce traitement que sont mis à jour différents collecteurs statistiques et compteurs d'observations. Ainsi le temps de séjour du client et son temps d'attente sont mémorisés. Le compteur du nombre de clients qui abandonnent en attente est mis à jour.

La classe *NbCustomersInSystem* est un événement technique c'est-à-dire un événement que l'on ne retrouve pas dans le système réel. Il s'agit d'une astuce de programmation afin d'exécuter un traitement. Son rôle est de capturer le nombre de clients présents dans le système à un instant $t$ de la simulation. Cette observation est alors mémorisée permettant par la suite de réaliser un graphe représentant la répartition du nombre de clients dans le système durant la simulation.

La classe *EndOfSim* est un événement technique. Dès que le simulateur rencontre cet événement, la simulation s'arrête.

La classe *Customer* représente un client. Il s'agit d'une structure de données composée de différentes variables décrivant un client. Ces variables peuvent être classées en trois catégories à savoir les données d'entrées, les variables de travail et les données de sorties. Les données d'entrées d'un client regroupent le temps d'arrivée d'un client, la taille du fichier que le client téléchargera, son impatience lorsqu'il téléchargera son fichier, son impatience quand il se trouve en file d'attente et la bande passante de téléchargement maximale dont il dispose. Les variables de travail sont utilisées durant la simulation afin de suivre l'évolution du client. Ces variables sont nécessaires au serveur pour gérer correctement le *Processor Sharing*. Elles se composent de la vitesse actuelle de téléchargement du client, de la taille de fichier restant à télécharger, du temps écoulé depuis l'événement précédent pour ce client. Enfin les variables de sortie expriment un résultat obtenu pour le client. Nous retrouvons dans cette catégorie le temps de séjour du client ainsi que son temps d'attente et son temps de service.

La classe *VideoSystem* est responsable de l'initialisation des différents composants que

sont l'horloge, les générateurs de nombres aléatoires, le serveur et la file d'attente. La méthode *ReadNextData()* est capable de générer un nouveau client possédant toutes ces données d'entrées remplies c'est-à-dire son temps d'arrivée, la taille de fichier à télécharger, son impatience en téléchargement, son impatience en attente et sa bande passante. De cette manière, nous pouvons générer des clients à la demande. C'est cette méthode qu'emploie l'événement *Arrival* afin d'obtenir un nouveau client pour planifier l'arrivée d'un nouveau client.

La classe *StatCollectors* est responsable de la collecte d'observations statistiques. Elle permet également la sérialisation d'une simulation sous forme d'un fichier *XML*. Cette classe est également le point d'entrée du simulateur. Cette classe dispose de quatre collecteurs de statistiques et de cinq compteurs d'observations. Nous avons un collecteur statistiques qui mémorise les temps de services des clients, un collecteur qui mémorise leurs temps d'attentes, un collecteur qui mémorise leurs temps de séjour et enfin un collecteur qui mémorise le nombre de clients présents à un instant t dans le système. Les compteurs d'observations enregistrent respectivement le nombre d'exclusions de clients, le nombre de clients servis, le nombre de clients qui abandonnent en téléchargement, le nombre de clients qui abandonnent en attente et le nombre de clients total.

L'interface *VideoServer* définit un ensemble de méthodes nécessaires au simulateur pour interagir avec le serveur. Toute classe fournissant une implémentation à ces méthodes peut devenir un serveur utilisable par le simulateur. Voici une explication des différentes méthodes de l'interface :

- La méthode *getSize()* renvoie le nombre de clients en service.
- La méthode *getMaxBandwidthServer()* renvoie la bande passante maximale du serveur pour l'envoi des fichiers vidéos.
- La méthode *rescheduleDepartures()* gère la stratégie du serveur lors de l'arrivée d'un client en service ou le départ d'un client du service.
- La méthode *addCustomer()* ajoute un client en service.
- La méthode *removeCustomer()* retire un client du service.
- La méthode *velocityCalculation* calcule la vitesse de téléchargement actuelle du client suivant la stratégie du serveur.

L'intérêt de cette interface est donc de pouvoir définir différents serveurs vidéos ayant une politique pour la mise en service des clients différente. Par exemple, le serveur peut définir une stratégie de mise en service des clients basée sur un algorithme de *Processor Sharing* ou encore définir une stratégie de mise en service basée sur mécanisme de priorités entre les clients. Suivant la stratégie choisie, une classe peut fournir une implémentation différente à ces méthodes.

L'interface *VideoQueue* définit un ensemble de méthodes nécessaires au simulateur pour interagir avec la file d'attente. Toute classe fournissant une implémentation à ces méthodes peut devenir une file d'attente utilisable par le simulateur. Voici une explication des différentes méthodes de l'interface :

- La méthode *getSize()* renvoie le nombre actuel de clients en attente.
- La méthode *getNbMaxCustomers()* renvoie le nombre maximum de clients autorisés dans

la file d'attente.
- La méthode *addCustomer()* ajoute un client dans la file d'attente.
- La méthode *retrieveCustomer()* retire un client de la file d'attente et le renvoie
- La méthode *removeCustomer()* supprime un client de la file d'attente.

L'intérêt de cette interface est donc de pouvoir définir différentes files d'attentes avec des disciplines de gestion différentes (FIFO, LIFO, ...).

La classe *VideoServerProcessorSharing* représente le serveur dans le système de distribution vidéo. Cette classe possède comme attribut la bande passante maximale dont dispose le serveur pour envoyer les fichiers vidéos aux différents clients en service. Cette classe implémente l'interface *VideoServer* c'est-à-dire qu'elle fournit une implémentation aux différentes méthodes de cette interface. La stratégie adoptée par ce serveur est le *Processor Sharing* tel que défini dans la section 4.3.3 (Calcul des départs).

La classe *VideoQueueFIFO* représente la file d'attente du système. Elle contient la borne maximum de clients présents dans la file. Cette classe implémente l'interface *VideoQueue*. Elle fournit donc une implémentation à toutes les méthodes de l'interface. La discipline de la file d'attente implémentée par cette classe est la discipline FIFO.

## 5.3 Interface graphique

Dans cette section, nous discutons de l'interface graphique réalisée ainsi que des choix technologiques faits. Même si le simulateur peut fonctionner parfaitement sans interface graphique dans un environnement de type console, celle-ci facilite l'accès au simulateur pour les utilisateurs.

**Choix technologique : librairie graphique** Le choix de la technologie utilisée pour réaliser l' interface graphique s'est porté naturellement vers la librairie graphique standard de Java qui est *Java Swing*. Étant donné que le simulateur est écrit en Java, le fait d'utiliser cette librairie n'ajoute pas d'exigences ou de dépendances supplémentaires. De plus, cette librairie possède l'avantage d'être portable. Les composants de cette librairie ont été écrits directement en Java et ne font pas appel aux composants graphiques du système d'exploitation. Il en résulte que cette interface n'est pas limitée à un type de système d'exploitation. La seule contrainte est l'installation de la plateforme Java.

**Choix technologique : Thread** Java propose un mécanisme appelé *Thread* permettant à une application d'exécuter plusieurs processus ayant chacun leurs propres ressources. Il faut savoir que la simulation est un travail consommateur en ressources. Si celui-ci est exécuté dans le même *thread* ou processus que l'interface graphique, celle-ci n'aura pas les ressources nécessaires pour réagir correctement aux interactions des utilisateurs. C'est pourquoi, nous avons développé une classe permettant de lancer une simulation dans un processus séparé de l'interface graphique. Ainsi l'interface graphique dispose des ses propres ressources et le simula-

teur également. Ceci peut permettre également de lancer plusieurs simulations simultanément. Nous n'avons cependant pas explorer cette voie pour l'instant.

**Description de l'interface graphique** L'interface graphique réalisée a été découpée en trois grandes fonctionnalités à savoir la réalisation d'une nouvelle simulation, la comparaison de mesures de performances et de graphiques issus de simulations enregistrées et un menu de préférences. Nous ne donnons ici qu'un aperçu général de l'interface graphique et de ses fonctionnalités. Pour plus détails à ce sujet, référez-vous au guide de l'utilisateur (voir Annexe A).

Sur la figure 5.5, nous voyons la fenêtre créée pour réaliser une nouvelle simulation :

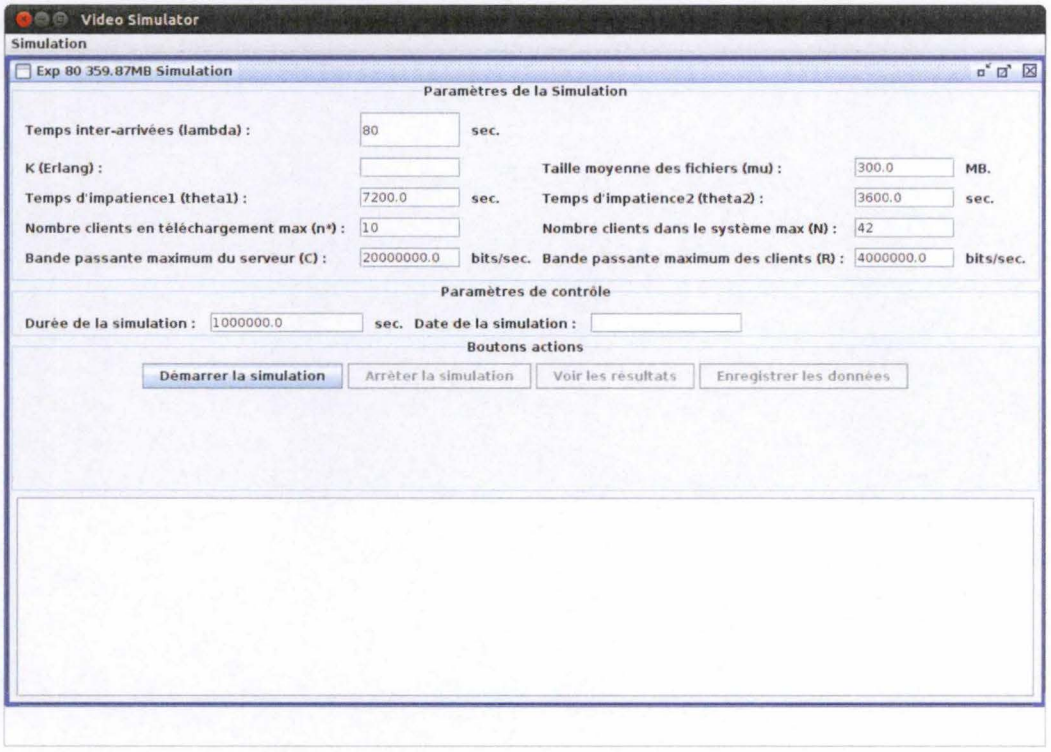
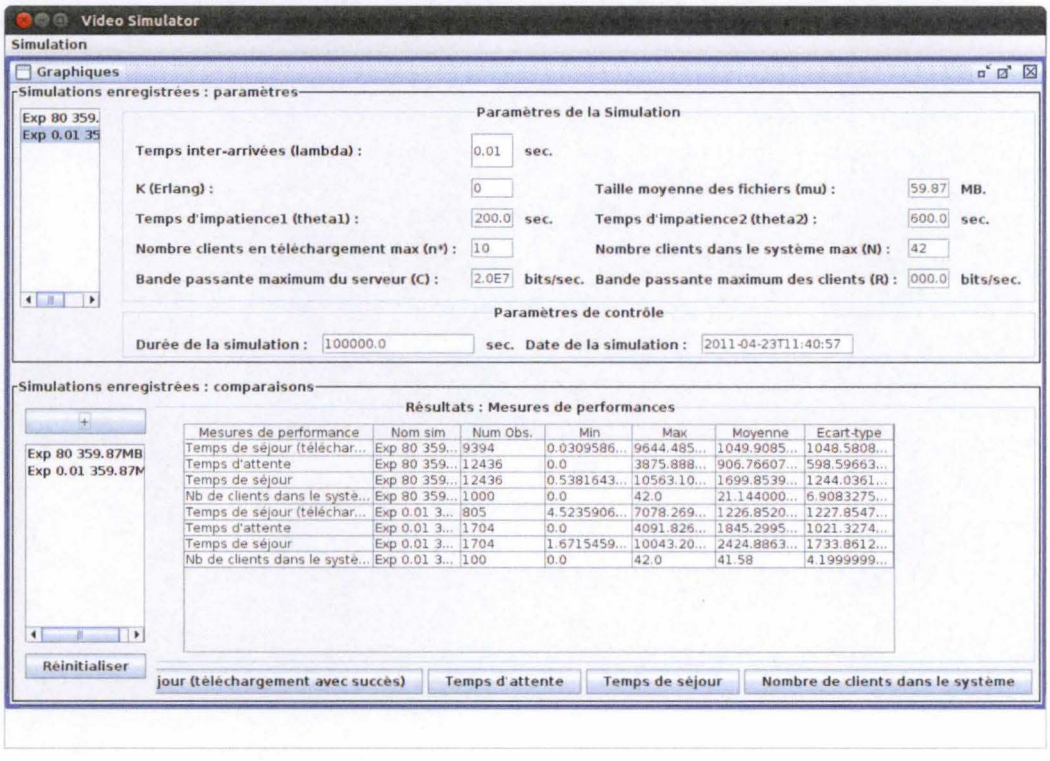



FIGURE 5.5 – Interface graphique : réaliser une nouvelle simulation

Comme nous le voyons sur la figure 5.5, la fenêtre est divisée en quatre zones. La zone "paramètres de simulation" a pour but de régler les différents paramètres d'entrée de la simulation. Une zone "paramètres de contrôle" permet de définir le temps de simulation. La zone "Boutons actions" permet de démarrer et arrêter une simulation. Elle permet également, une fois la simulation terminée de voir les résultats de la simulation c'est-à-dire les différentes mesures de performances et graphiques calculés. Le bouton "enregistrer les données" permet de sauvegarder les données de la simulation sous la forme d'un fichier XML (voir section 5.4). Les différents résultats de la simulation peuvent alors être revus ou comparés avec d'autres simulations. Enfin la dernière zone en bas de la fenêtre affiche le déroulement de la simulation dès son démarrage. Une simulation pouvant durer un temps assez long, cette zone permet alors de savoir où l'on se situe au niveau de la durée de la simulation.

Voici la fenêtre créée pour comparer des simulations enregistrées au préalable :




| Mesures de performance | Nom sim | Num Obs. | Min | Max | Moyenne | Ecart-type |
|---|---|---|---|---|---|---|
| Temps de séjour (téléchar... | Exp 80 359... | 9394 | 0.0309586... | 9644.485... | 1049.9085... | 1048.5808... |
| Temps d'attente | Exp 80 359... | 12436 | 0.0 | 3875.888... | 906.76607... | 598.59663... |
| Temps de séjour | Exp 80 359... | 12436 | 0.5381643... | 10563.10... | 1699.8539... | 1244.0361... |
| Nb de clients dans le systè... | Exp 80 359... | 1000 | 0.0 | 42.0 | 21.144000... | 6.9083275... |
| Temps de séjour (téléchar... | Exp 0.01 3... | 805 | 4.5235906... | 7078.269... | 1226.8520... | 1227.8547... |
| Temps d'attente | Exp 0.01 3... | 1704 | 0.0 | 4091.826... | 1845.2995... | 1021.3274... |
| Temps de séjour | Exp 0.01 3... | 1704 | 1.6715459... | 10043.20... | 2424.8863... | 1733.8612... |
| Nb de clients dans le systè... | Exp 0.01 3... | 100 | 0.0 | 42.0 | 41.58 | 4.1999999... |

FIGURE 5.6 – Interface graphique : comparer des simulations

La fenêtre présente sur la figure 5.6 permet de comparer des simulations enregistrées. Celle-ci est découpée en deux zones à savoir une zone "paramètres" et une zone "comparaisons". La zone "paramètres" affiche tout simplement les paramètres avec lesquels une simulation a été réalisée. Nous pouvons sélectionner dans la partie gauche une simulation et ces différents paramètres s'afficheront. La zone "comparaison" affiche les mesures de performances des simulations sélectionnées au préalable sous la forme d'un tableau. Les résultats graphiques sont également disponibles via les quatre boutons présents en bas de la fenêtre.

## 5.4 Mécanismes de performances et d'efficacité appliqués au simulateur

Dans cette section, nous présentons les différents mécanismes utilisés afin de rendre le simulateur efficace et performant. Les simulations à événements discrets doivent s'exécuter un certain temps afin d'arriver à un état stationnaire (voir 2.3.2). Il n'est pas rare de devoir exécuter une simulation une journée entière avant d'obtenir des résultats. Ceci implique donc une consommation importante en temps processeur, en mémoire mais aussi en temps d'attente pour l'utilisateur.

**Approche *Next-Event*** L'approche *Next-Event* offre deux mécanismes de performances.

Premièrement, l'horloge dans une approche *Next-Event* avance de manière asynchrone c'est-à-dire d'événement en événement. Cette horloge n'avance pas de manière séquentielle avec le temps. Ceci veut dire que si aucun événement ne doit survenir pendant dix minutes, nous n'attendons pas dix minutes mais nous exécutons directement l'événement qui doit avoir lieu après ces dix minutes. Il en résulte un gain dans le temps d'attente de l'utilisateur.

Deuxièmement, l'approche *Next-Event* ne doit connaitre que les prochains événements. Si l'on reprend l'algorithme de l'approche *Next-Event* (voir 4.2.2), nous remarquons que l'algorithme traite l'événement courant, ensuite génère les prochains événements qui eux-mêmes seront traités et ainsi de suite. Cet algorithme se déplace d'événements en événements. Il en ressort que nous ne devons pas connaitre tous les événements à l'avance ce qui constituerait une occupation mémoire importante. Les prochains événement doivent seulement être connus. De plus dès qu'un un événement est traité, il est oublié et la mémoire consommée par cet événement est libérée.

**Sauvegarde des simulations** Les sauvegardes des simulations peuvent être considérées comme un mécanisme de performance pour plusieurs raisons.

Premièrement, les simulations par événements discrets sont consommatrices en temps. Pour rappel, une simulation doit arriver à un état stationnaire et celui-ci ne peut être atteint qu'après un temps assez conséquent (voir section 2.3.2). Il est donc utile de conserver le résultat d'une simulation afin de ne pas devoir la réitérer et épargner le temps d'attente de l'utilisateur.

Deuxièmement, bien que s'appuyant sur des générateurs de nombres aléatoires, une simulation par événements discrets avec les mêmes paramètres en entrée produit les mêmes résultats en sortie. La séquence de nombres aléatoires générée avec les mêmes paramètres en entrée renverra la même séquence de nombres aléatoires (voir 2.3.1). L'algorithme du simulateur est quant à lui déterministe. Par conséquent, il est intéressant de mémoriser le résultat d'une simulation puisqu'avec les mêmes paramètres son résultat restera identique.

Troisièmement, l'analyse des résultats s'effectue principalement par des comparaisons entre plusieurs simulations. La sauvegarde de résultats permet donc de comparer instantanément différentes simulations déjà réalisées.

Pour réaliser ces sauvegardes, plusieurs méthodes existent. Nous avons choisi d'enregistrer les données des simulations sous un format *XML*. Ce format présente plusieurs avantages. Il est facilement lisible et compréhensible pour les êtres humains. Des outils de validation existent permettant de s'assurer que le fichier possède un format correct. Enfin, les fichiers de sauvegarde possèdent une taille assez importante. Il faut donc un mécanisme de lecture assez performant pour que le chargement soit relativement rapide. La consommation mémoire doit aussi être la plus minime possible. Le format *XML* dispose de deux *parser* standards qui sont *Document Object Model* (DOM) et *Simple API for XML* (SAX). La méthode *DOM* charge entièrement le fichier en mémoire et permet de naviguer dans le fichier de n'importe quelle manière. Cette méthode est généralement employée pour la modification d'un fichier XML. Cependant, elle est coûteuse en ressources mémoires.

La méthode *SAX* est basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML. Un événement est par exemple la rencontre d'une balise XML. Une application utilisant SAX implémente généralement des gestionnaires d'événements, lui permettant d'effectuer des opérations selon le type d'élément rencontré. L'avantage de cette méthode est qu'il n'est pas nécessaire de charger l'entièreté du fichier en mémoire. Nous avons donc opté pour l'emploi de cette méthode. En annexe se trouve le schéma de validation XML que nous avons créé ainsi qu'un exemple de sauvegarde d'une simulation. (voir annexe B).

# Chapitre 6

# Vérification et Validation du simulateur

## Sommaire


**6.1 Vérifier le simulateur** . . . . . **63**
**6.2 Valider le simulateur** . . . . . **64**


Nous reprenons dans ce chapitre les étapes 5 et 6 de la méthodologie de la simulation par événements discrets (voir section 3.1). La section 6.1 décrit les techniques utilisées afin de s'assurer du comportement correct du simulateur. Ensuite la section 6.2 explique les mécanismes employés afin de valider le simulateur c'est-à-dire s'assurer que le système simulé correspond au système réel.

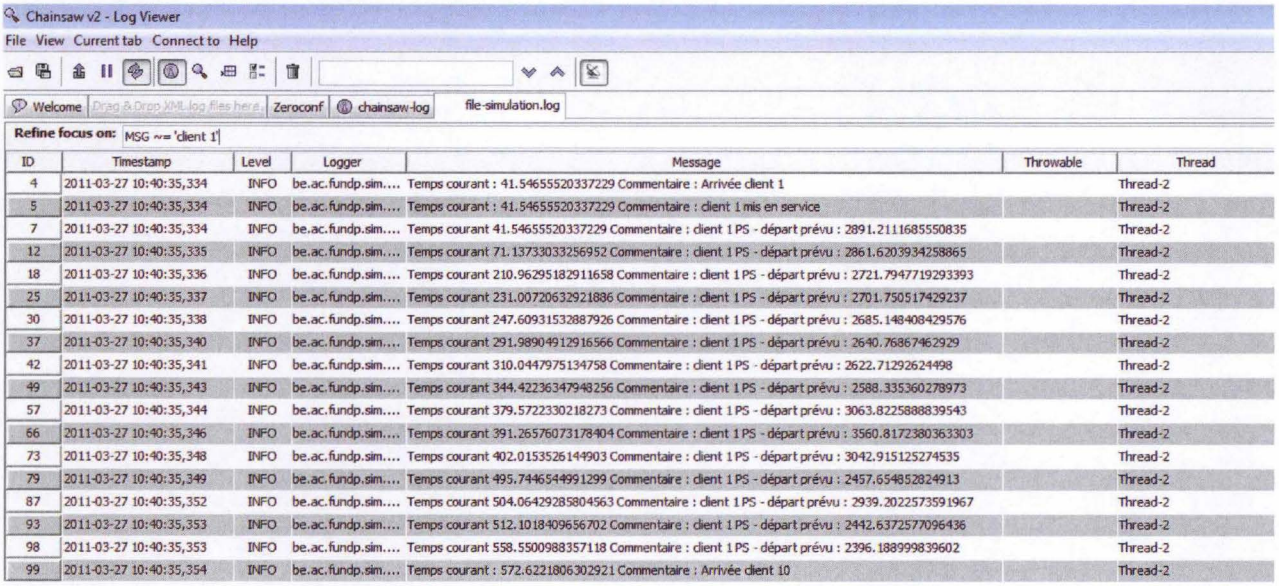
## 6.1 Vérifier le simulateur

La vérification du modèle s'effectue essentiellement par des tests. Il s'agit ici de vérifier que le code est correct et se comporte de la manière souhaitée. A cette fin nous avons utilisé la librairie *log4j*[1] qui nous a permis de tracer le comportement du simulateur. Le choix s'est porté sur cette librairie car elle offre une architecture souple pour gérer les traces. Cette librairie est configurable via un fichier texte extérieur à l'application. Nous pouvons dès lors grâce à ce fichier activer ou désactiver les traces sans devoir modifier le simulateur. Cette librairie permet aussi de définir différents *Appender* c'est-à-dire différentes sorties pour les traces. Ces sorties peuvent être un fichier texte, une console ou un composant écrit par nos soins. Nous avons développé une console *Java Swing* afin de visualiser ces traces directement depuis l'interface graphique du simulateur.

La librairie *Log4j* devient un standard pour gérer les traces. Dès lors de nombreux logiciels se sont développés afin de pouvoir lire ces fichiers de traces et de pouvoir y effectuer des traitements. Vu la taille importante que peut avoir ces fichiers de traces, la lecture n'est pas toujours simple. Nous avons eu recours notamment au logiciel *Chainsaw* afin de filtrer le fichier de trace. Grâce à ce logiciel nous pouvons par exemple filtrer le fichier de trace pour

1. `http://logging.apache.org/log4j/`

n'avoir que les informations liées au client numéro 1.



Chainsaw v2 - Log Viewer

File View Current tab Connect to Help

Welcome | Drag & Drop XML log files here | Zeroconf | chainsaw-log | file-simulation.log

Refine focus on: MSG ~= 'client 1'

| ID | Timestamp | Level | Logger | Message | Throwable | Thread |
|---|---|---|---|---|---|---|
| 4 | 2011-03-27 10:40:35,334 | INFO | be.ac.fundp.sim.... | Temps courant : 41.54655520337229 Commentaire : Arrivée client 1 | | Thread-2 |
| 5 | 2011-03-27 10:40:35,334 | INFO | be.ac.fundp.sim.... | Temps courant : 41.54655520337229 Commentaire : client 1 mis en service | | Thread-2 |
| 7 | 2011-03-27 10:40:35,334 | INFO | be.ac.fundp.sim.... | Temps courant 41.54655520337229 Commentaire : client 1 PS - départ prévu : 2891.2111685550835 | | Thread-2 |
| 12 | 2011-03-27 10:40:35,335 | INFO | be.ac.fundp.sim.... | Temps courant 71.13733033256952 Commentaire : client 1 PS - départ prévu : 2861.6203934258865 | | Thread-2 |
| 18 | 2011-03-27 10:40:35,336 | INFO | be.ac.fundp.sim.... | Temps courant 210.96295182911658 Commentaire : client 1 PS - départ prévu : 2721.7947719293393 | | Thread-2 |
| 25 | 2011-03-27 10:40:35,337 | INFO | be.ac.fundp.sim.... | Temps courant 231.00720632921886 Commentaire : client 1 PS - départ prévu : 2701.750517429237 | | Thread-2 |
| 30 | 2011-03-27 10:40:35,338 | INFO | be.ac.fundp.sim.... | Temps courant 247.60931532887926 Commentaire : client 1 PS - départ prévu : 2685.148408429576 | | Thread-2 |
| 37 | 2011-03-27 10:40:35,340 | INFO | be.ac.fundp.sim.... | Temps courant 291.98904912916566 Commentaire : client 1 PS - départ prévu : 2640.76867462929 | | Thread-2 |
| 42 | 2011-03-27 10:40:35,341 | INFO | be.ac.fundp.sim.... | Temps courant 310.0447975134758 Commentaire : client 1 PS - départ prévu : 2622.71292624498 | | Thread-2 |
| 49 | 2011-03-27 10:40:35,343 | INFO | be.ac.fundp.sim.... | Temps courant 344.42236347948256 Commentaire : client 1 PS - départ prévu : 2588.335360278973 | | Thread-2 |
| 57 | 2011-03-27 10:40:35,344 | INFO | be.ac.fundp.sim.... | Temps courant 379.5722330218273 Commentaire : client 1 PS - départ prévu : 3063.8225888839543 | | Thread-2 |
| 66 | 2011-03-27 10:40:35,346 | INFO | be.ac.fundp.sim.... | Temps courant 391.26576073178404 Commentaire : client 1 PS - départ prévu : 3560.8172380363303 | | Thread-2 |
| 73 | 2011-03-27 10:40:35,348 | INFO | be.ac.fundp.sim.... | Temps courant 402.01535261449O3 Commentaire : client 1 PS - départ prévu : 3042.915125274535 | | Thread-2 |
| 79 | 2011-03-27 10:40:35,349 | INFO | be.ac.fundp.sim.... | Temps courant 495.7446544991299 Commentaire : client 1 PS - départ prévu : 2457.654852824913 | | Thread-2 |
| 87 | 2011-03-27 10:40:35,352 | INFO | be.ac.fundp.sim.... | Temps courant 504.06429285804563 Commentaire : client 1 PS - départ prévu : 2939.2022573591967 | | Thread-2 |
| 93 | 2011-03-27 10:40:35,353 | INFO | be.ac.fundp.sim.... | Temps courant 512.1018409656702 Commentaire : client 1 PS - départ prévu : 2442.6372577096436 | | Thread-2 |
| 98 | 2011-03-27 10:40:35,353 | INFO | be.ac.fundp.sim.... | Temps courant 558.5500988357118 Commentaire : client 1 PS - départ prévu : 2396.188999839602 | | Thread-2 |
| 99 | 2011-03-27 10:40:35,354 | INFO | be.ac.fundp.sim.... | Temps courant : 572.6221806302921 Commentaire : Arrivée client 10 | | Thread-2 |

FIGURE 6.1 – Analyse du fichier log par l'outil *Chainsaw* pour le client 1

Un autre mécanisme a été utilisé en combinaison avec les fichiers de traces. Il s'agit de la possibilité de donner au simulateur un jeu de données d'entrée réduit où les informations ne sont plus aléatoires. Nous pouvons fixer les différents paramètres donnés au simulateur comme le temps d'arrivée des clients, la taille des fichiers à télécharger et les différents types d'impatiences liées aux clients. Vu que les paramètres sont fixés, les résultats sont connus à l'avance et peuvent être comparés facilement avec le fichier de trace obtenu par le simulateur. Un exemple de jeu de données d'entrée est présent en annexe C.

## 6.2 Valider le simulateur

La validation du modèle consiste à confronter les résultats obtenus via le programme avec ceux du système étudié. Cette validation commence au préalable par quelques tests de cohérence. Le nombre de départ du à l'impatience augmente-t-il si le nombre de clients augmente ? Le nombre d'exclusion augmente-t-il si le nombre de clients augmente ?

L'étude [5] a étudié le système de distribution de contenu vidéo selon une approche mathématique. Des graphiques ont été publiés par cette étude. Nous pouvons donc confronter les résultats du simulateur avec les graphiques de cette étude. Nous décrivons ci-dessous les scénarios imaginés par l'étude et qui ont conduit à la réalisation de graphiques. Nous les confrontons ensuite aux graphiques du simulateur.

**Scénarios** Pour le premier scénario, les paramètres ont été définis comme suit :

- le taux d'arrivée des clients s'effectue à un rythme moyen de $\lambda = 1/0.01$ secondes.

- la taille moyenne des fichiers $\mu^{-1}$ a été fixée à 359.87 MegaBytes.
- la bande passante du serveur $C$ a été fixée à 20 Mbps.
- la bande passante des client $R$ a été fixée à 4 Mbps.
- le nombre maximum de clients en téléchargement est fixé à 10.
- le nombre maximum de clients dans le système est fixé à 20.
- la durée moyenne d'impatience pour les clients en téléchargement $\theta_1^{-1}$ est fixée à 2 heures.
- la durée moyenne d'impatience pour les clients en attente $\theta_2^{-1}$ est fixée à 1 heure.

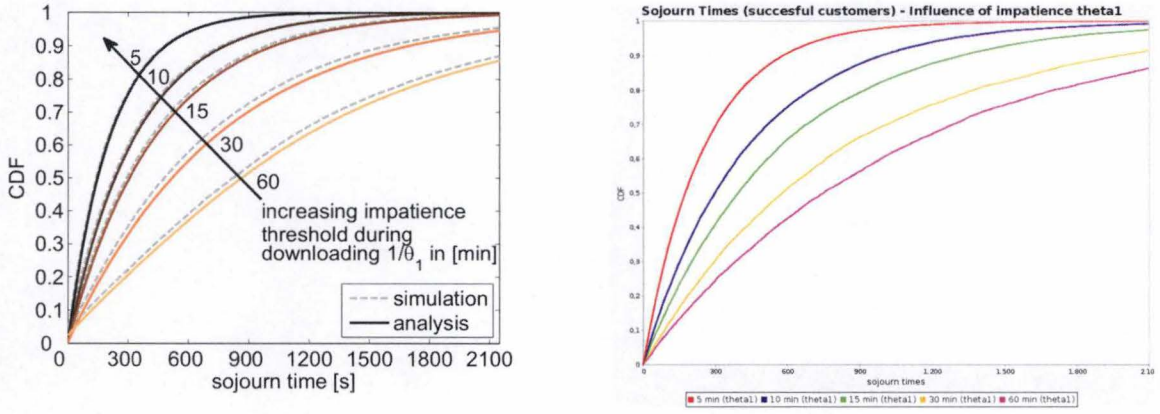



FIGURE 6.2 – Comparaison entre le graphique Fig2a de l'étude [5] et le graphique issu du simulateur

Sur la figure 6.2, nous avons à gauche pour le scénario défini ci-dessus le graphique réalisé par l'étude et à droite le graphique issu du simulateur. Nous remarquons une similitude assez forte entre ces graphiques ce qui nous conforte dans l'idée que le simulateur simule correctement le système étudié.

Pour le deuxième scénario imaginé, les paramètres on été définis comme suit :

- le taux d'arrivée des clients s'effectue à un rythme moyen de $\lambda = 1/80$ secondes.
- la taille moyenne des fichiers $\mu^{-1}$ a été fixée à 359.87 MegaBytes.
- la bande passante du serveur $C$ a été fixée à 20 Mbps.
- la bande passante des client $R$ a été fixée à 4 Mbps.
- le nombre maximum de clients en téléchargement est fixé à 10.
- le nombre maximum de clients dans le système est fixé à 42.
- la durée moyenne d'impatience pour les clients en téléchargement $\theta_1^{-1}$ est fixée à 2 heures.
- la durée moyenne d'impatience pour les clients en attente $\theta_2^{-1}$ est fixée à 1 heure.

Sur la figure 6.3, nous avons à gauche pour le scénario défini ci-dessus le graphique réalisé par l'étude et à droite le graphique issu du simulateur. Nous remarquons également une similitude assez forte entre ces graphiques.

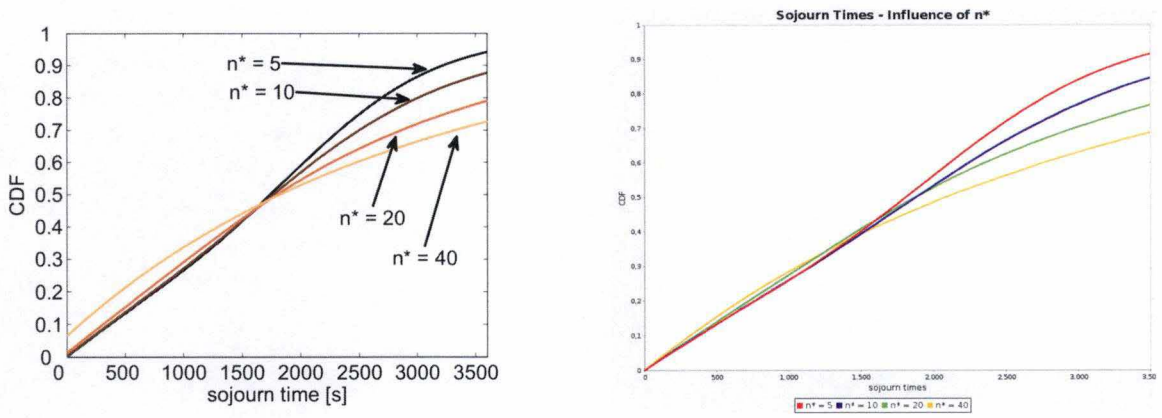



FIGURE 6.3 – Comparaison entre le graphique Fig4a de l'étude [5] et le graphique issu du simulateur

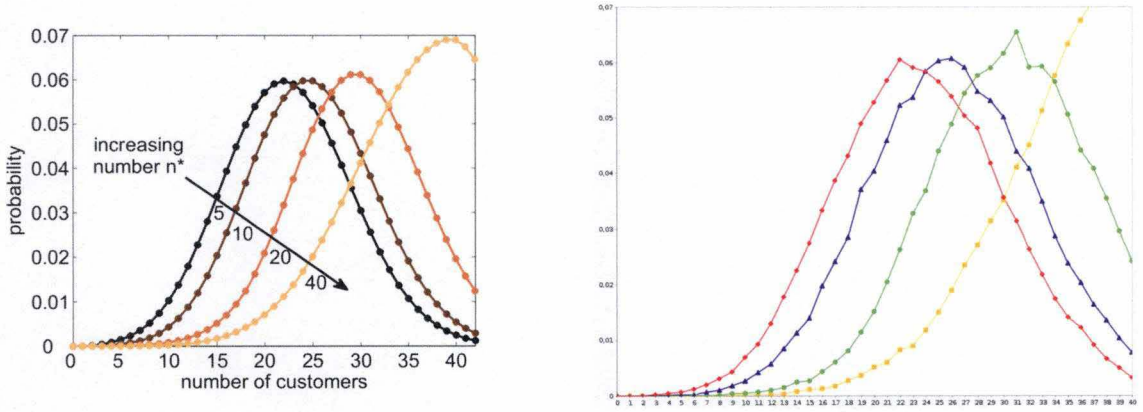



FIGURE 6.4 – Comparaison entre le graphique Fig4b de l'étude [5] et le graphique issu du simulateur

Sur la figure 6.4, nous avons à gauche pour le scénario défini ci-dessus le graphique réalisé par l'étude et à droite le graphique issu du simulateur. Nous remarquons également une similitude entre ces graphiques.

**Conclusion** Bien qu'il n'est pas évident de comparer précisément des graphiques, nous pouvons remarquer une similitude assez forte entre les graphiques du simulateur et ceux de l'étude. Ceux-ci sont construits à partir de cinq simulations différentes où un seul paramètre est changé. Nous observons que les graphiques issus du simulateur se rapprochent fortement des graphiques de l'étude et ceci pour plusieurs scénarios ayant des paramètres différents. Nous pouvons donc admettre que le système de l'étude et le système simulé sont identiques et que le simulateur est valide.

# Chapitre 7

# Modification du modèle conceptuel

## Sommaire


**7.1 Modifications apportées . . . . . 67**
**7.2 Étude numérique . . . . . 70**
7.2.1 Présentation des scénarios . . . . . 70
7.2.2 Résultats des simulations . . . . . 71
**7.3 Conclusion . . . . . 79**


Dans ce chapitre, nous nous situons à l'étape "Prendre des décisions" de la méthodologie de la simulation par événements discrets expliquée dans la section 3.1. Une hypothèse sur le modèle conceptuel a été modifiée. La taille des fichiers téléchargés par les clients a été jusqu'ici définie par une loi exponentielle. Nous allons remplacer cette loi exponentielle par une loi Erlang. Nous expliquons dans ce chapitre cette modification, la raison de cette modification, ses implications ainsi que les résultats obtenus.

## 7.1 Modifications apportées

Nous avons vu dans la section 3.2.2 que le modèle conceptuel du simulateur fait l'hypothèse que la taille des fichiers téléchargés par les clients est distribuée selon une loi exponentielle. Nous avons vu également qu'une loi Erlang se rapprochait plus de la réalité qu'une loi exponentielle (voir section 3.2.3 ainsi que la figure 3.2).

La décision de confronter les résultats effectués avec une distribution exponentielle d'une part et les résultats effectués avec une distribution erlang d'autre part, pour la distribution de la taille des fichiers a été prise.

Nous présentons ci-dessous la distribution Erlang ainsi que ces particularités par rapport à l'exponentielle.

**Définition de l'Erlang** Voici une définition de la loi Erlang et de sa relation avec la loi Exponentielle.

**Définition de l'Erlang 1.**

Une variable aléatoire continue $X$ est une *Erlang*(n,b) si et seulement si

$X = X_1 + X_2 + ... + X_n$,

avec $X_1, X_2, ..., X_n$ une séquence d'exponentielle(b) indépendantes, $n \in \mathbb{N}, b \in \mathbb{R}$.

Suivant cette définition, nous pouvons remarquer qu'une distribution Erlang possède deux paramètres d'entrée contrairement à la distribution Exponentielle qui n'en possède qu'un seul. En fait l'exponentielle est un cas particulier de la distribution Erlang où $n = 1$.

**Fonction inverse de l'Erlang** Nous avons vu dans la section 2.3.1 que la génération des variables aléatoires utilise la fonction inverse de distribution afin de générer une variable aléatoire respectant une loi de probabilité. La fonction inverse de l'Erlang ne peut pas être calculée de la même manière que l'exponentielle. Nous savons (voir section 2.3.1) que la fonction de distribution inverse est liée à la fonction de distribution cumulative. Nous présentons ci-dessous selon une approche mathématique le lien entre ces deux fonctions. Ceci explique aussi pourquoi l'approche d'inversion directe n'est pas possible avec la distribution Erlang.

**Définition de la fonction de distribution cumulative(FDC) de l'exponentielle 2.**

Soit $X$ une variable aléatoire exponentielle de paramètre $\mu$,

alors la fonction de distribution cumulative (FDC) est la suivante : $F(x) = 1 - e^{(-x/\mu)}$.

La fonction de distribution inverse se calcule alors en résolvant l'équation : $u = F(x)$ ce qui donne $x = F^{-1}(u) = -\mu\ ln(1-u) \qquad 0 < u < 1$

Pour l'Erlang, nous avons la fonction de distribution cumulative suivante :

**Définition de la fonction de distribution cumulative(FDC) de l'Erlang 3.**

Soit $X$ une variable aléatoire Erlang(n,b),

alors la fonction de distribution cumulative (FDC) est la suivante : $F(x) = 1 - [e^{(-\frac{x}{b})}](\sum_{i=0}^{n-1} \frac{(x/b)^i}{i!})$.

La fonction de distribution inverse de la distribution Erlang ne peut se calculer de manière directe en résolvant l'équation : $u = F(x)$. D'autres approches sont alors utilisées. Nous donnons ici l'idée générale de ces approches.

Vu qu'une distribution Erlang est une somme de distributions exponentielles, une première approche consiste à additionner $n$ variables aléatoires exponentielles. Cependant cet algorithme a une complexité en $O(n)$ et s'avère donc peu performant lorsque $n$ est grand.

Une deuxième approche consiste à utiliser une fonction approximant le plus précisément possible la fonction inverse.

Enfin la dernière approche, qui est aussi la plus répandue, consiste à résoudre l'équation numériquement. Comme l'équation $u = F(x)$ ne peut pas se résoudre algébriquement de manière simple, l'idée est de se rapprocher de cette valeur par approximation. Plusieurs algorithmes sont disponibles pour cette catégorie. Nous avons utilisé l'algorithme recommandé par la librairie *SSJ* à savoir celui de la méthode de *Newton* qui possède l'avantage d'un bon compromis entre convergence et efficacité. Voici une définition de cet algorithme :

**Définition de l'algorithme d'inversion de Newton 4.**

Soit un $u \in (0, 1)$, la fonction de distribution cumulative F(t), la fonction de densité de probabilité f(t) qui est la dérivée première de la fonction de distribution cumulative et un paramètre de convergence $e > 0$, l'algorithme de *Newton* peut résoudre l'équation $x = F^{-1}(u)$ de la manière suivante :

$x = \mu$
do {
    $t = x$;
    $x = t + (u - F(t))/f(t)$;
} while ($|x - t| > e$);
return x;

Cet algorithme utilise les séries de *Taylor*. Il s'agit de séries construites à partir des dérivées successives d'une fonction. Voici la définition des séries des *Taylor* :

**Définition des séries de *Taylor* 5.**

$$F(x) = F(t) + F'(t)(x - t) + \frac{1}{2!}F''(t)(x - t)^2 + ....$$

Au plus le degré du polynôme de cette série est élevé, au plus nous nous rapprochons de la véritable fonction. Évidemment au plus le degré du polynôme est élevé, au plus le calcul est consommateur en temps. C'est pourquoi l'algorithme de Newton possède un paramètre de convergence qui stoppe l'algorithme une fois que la valeur approchée est suffisamment proche de la véritable fonction. Dans notre cas, remarquons aussi que la première dérivée de la fonction de distribution cumulative est la fonction de densité de probabilité. Pour plus de détails sur cet algorithme, veuillez vous référez à [3, Chapitre 7, page 297].

**Choix des paramètres de l'Erlang** Nous avons vu ci-dessus que la distribution Erlang possède deux paramètres d'entrée tandis que l'Exponentielle n'en possède qu'un seul. Dès lors, si nous voulons comparer des résultats de simulations faites avec la distribution exponentielle

avec des résultats de simulations faites avec la distribution Erlang, la relation entre ces paramètres doit être définie. La taille moyenne des fichiers sera donc définie soit par un paramètre dans le cas de l'exponentielle, soit par deux paramètres dans le cas d'une Erlang.

Nous savons d'après l'étude réalisée (voir section 3.2.3) qu'un échantillon a été calculé avec une taille moyenne des fichiers est de 368.31 MB et un écart-type de 196.82 MB. Nous avons décider d'utiliser cette moyenne comme paramètre pour l'exponentielle. Pour l'erlang, nous avons tenter de calculer un $n$ à partir de cette moyenne et écart-type. Ci-dessous la résolution d'un système d'équations à deux inconnues afin de trouver $n$ et $b$ à partir de l'échantillon.

Soit une Erlang(n,b) de moyenne 368.31 MB

Moyenne de l'Erlang : $\mu_1$
Ecart-type de l'Erlang : $e_1$
$m_1 = n/b$
$e_1 = \sqrt{n}/b$

$m_1 = 368.31MB$
$e_1 = 196.82MB$
$n = 368.31 * b$
$\sqrt{n} = 196.82 * b$
$\sqrt{368.31 * b} = 196.82 * b$
$368.31 * b = 196.82^2 * b^2$
$b/b^2 = 196.82^2/368.31$
$b = 368.31/192.86^2$
$b = 0.0095$
$n = 3.5$

Nous avons donc trouver une valeur pour $n$ de $3,5$. Comme le paramètre $n$ de la distribution Erlang doit être un entier, nous avons réaliser des simulations avec $n = 3$ et $n = 4$. Dans la suite de ce chapitre, nous avons uniquement conservé les simulations avec $n = 3$. Les résultats obtenus avec $n = 4$ possèdent les mêmes caractéristiques.

## 7.2 Étude numérique

### 7.2.1 Présentation des scénarios

Nous avons réalisé différentes simulations avec les mêmes paramètres afin de comparer les résultats obtenus sur les mesures de performances par une loi Exponentielle et une loi Erlang. Différents scénarios ont été imaginés, nous en avons retenu trois à savoir :

- scénario 1 : une situation de saturation du système
- scénario 2 : une situation intermédiaire
- scénario 3 : une situation considérée comme normale

Voici une explication des trois scénarios.

**Explication des scénarios** Pour les différents scénarios, les paramètres ont été définis comme suit :

- le paramètre k de l'Erlang est fixé à 3.
- la taille moyenne des fichiers $\mu^{-1}$ a été fixée à 368.31 MegaBytes.
- la bande passante du serveur $C$ a été fixée à 20 Mbps.
- la bande passante des clients $R$ a été fixée à 4 Mbps.
- le nombre maximum de clients en téléchargement est fixé à 10.
- la durée moyenne d'impatience pour les clients en téléchargement $\theta_1^{-1}$ est fixée à 2 heures.
- la durée moyenne d'impatience pour les clients en attente $\theta_2^{-1}$ est fixée à 1 heure.

Ensuite deux paramètres à savoir le taux d'arrivée des clients ainsi que le nombre maximum de clients dans le système doivent encore être définis. Nous avons fait varier ces deux paramètres ce qui a donné ces trois scénarios. Nous avons nommé ces trois scénarios sur base du taux d'exclusion c'est-à-dire le nombre de clients exclus par rapport au nombre total de clients. En effet, le scénario 1 possède un taux d'exclusion de clients élevé, le scénario 2 un taux d'exclusion intermédiaire et le scénario 3 un taux d'exclusion faible.

Pour le scénario 1, le nombre de clients maximum présents dans le système a été fixé à 20. Nous avons fixé le taux d'arrivée à 0.01 secondes ce qui constitue un taux d'arrivée relativement élevé. Il résulte de ce scénario un taux d'exclusion de clients élevé. Ce scénario représente une situation de saturation du système.

Pour le scénario 2, nous avons diminuer le taux d'arrivée des clients et augmenter le nombre maximum de clients présents dans le système. Ce nombre a été fixé à 42 et le taux d'arrivée des clients a été fixé à 30 secondes. Ce scénario représente une situation intermédiaire. Il résulte de ce scénario un taux d'exclusion moindre que pour le scénario 1.

Pour le scénario 3, nous avons encore diminuer le taux d'arrivée des clients. Nous avons fixé celui-ci à 80 secondes. Les clients arrivent donc à une débit très raisonnable. Il résulte de ce scénario un taux d'exclusion de clients assez faible. Nous avons défini ce scénario comme étant une situation considérée comme normale.

### 7.2.2 Résultats des simulations

Nous avons simulé les trois scénarios décrits ci-dessus. Les résultats des mesures de performances pour ces différents scénarios sont présentés ci-dessous sous la forme de tableaux et de graphiques. Une explication des titres et des colonnes des tableaux est donnée ci-dessous.

**Légende des tableaux de résultats** Voici une explication des titres de colonnes :

- Dist. : indique la distribution utilisée pour la taille des fichiers lors de simulation. Il s'agit soit de l'Exponentielle(Exp) ou de l'Erlang.
- $\mu$ Tps séj. : indique le temps de séjour (temps de téléchargement) moyen des clients. Cette valeur est exprimée en secondes.
- $\sigma$ Tps séj. : indique l'écart-type du temps de séjour. Cette valeur est exprimée en secondes.
- % excl. : indique le taux d'exclusion. Il s'agit du nombre d'exclusions par rapport au nombre total de clients.
- % abandon 1 : indique le taux d'abandon en téléchargement. Il s'agit du nombre de clients qui quittent le système alors qu'ils étaient occupés à télécharger un fichier par rapport au nombre total de clients.
- % abandon 2 : indique le taux d'abandon en attente. Il s'agit du nombre de clients qui quittent le système alors qu'ils se trouvaient dans la file d'attente par rapport au nombre total de clients.
- Nb cli. sys. : indique la moyenne du nombre de clients dans le système durant la simulation.

**Résultats Scénario 1** Le tableau ci-dessous présente les résultats des mesures de performances des simulations effectuées avec l'Exponentielle et l'Erlang pour le scénario 1.

| Dist. | $\mu$ Tps séj. | $\sigma$ Tps séj. | % excl. | % abandon 1 | % abandon 2 | Nb cli. sys. |
|---|---|---|---|---|---|---|
| Exp. | 1888,465 | 1397,603 | 0,99 | 0,000014 | 0,000027 | 19,9 |
| Erlang | 1875,865 | 1388,552 | 0.99 | 0,000013 | 0,000027 | 19,9 |

Pour cette situation de saturation du système, nous remarquons évidemment un taux d'exclusion de clients important. Le temps de séjour moyen est légèrement plus faible pour la distribution Erlang, les autres résultats sont similaires. Nous pouvons donc conclure pour ce scénario que les deux distributions sont semblables. Ceci se vérifie aussi sur la figure 7.1 représentant la répartition des temps de séjour respectivement pour l'exponentielle et pour l'erlang. Nous voyons sur cette figure que les deux distributions sont extrêmement proches.

Afin de confirmer cette interprétation nous avons réalisé un test d'hypothèse. Il s'agit du test d'égalité des moyennes nommé test de Student. Il se définit comme suit :

$H_0$ : l'hypothèse d'égalité des moyennes est acceptée
$H_1$ : l'hypothèse d'égalité des moyennes est rejetée
$\mu_1$ : moyenne de l'échantillon 1
$\mu_2$ : moyenne de l'échantillon 2
$\sigma_1$ écart-type de l'échantillon 1
$nobs_1$ : nombre d'observations de l'échantillon 1
$t*$ : valeur présente dans la table des fractiles de la distribution Student

$$t = \frac{\mu_1 - \mu_2}{\sigma_1/\sqrt{nobs_1}} < t*$$

Si $t < t*$ alors $H_0$ sinon $H_1$.

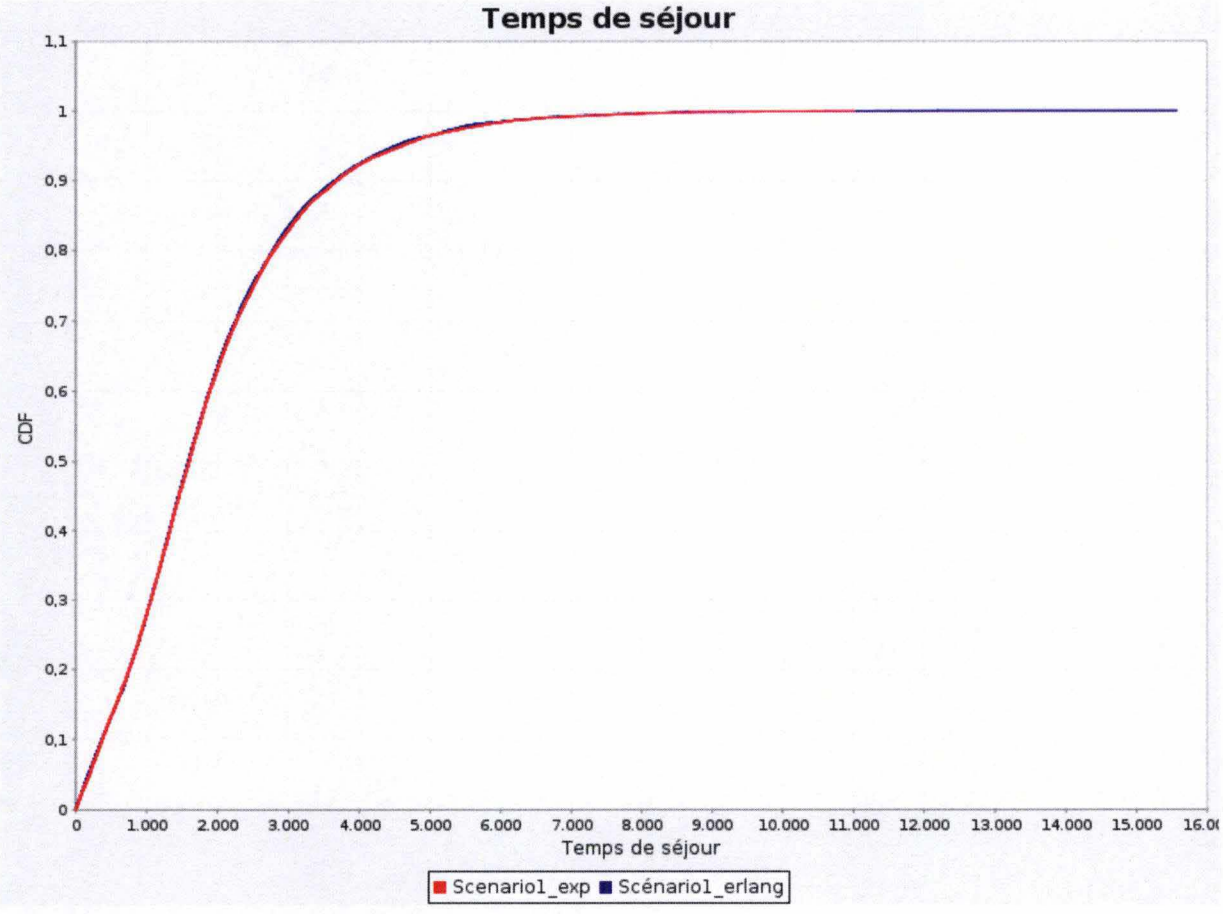



FIGURE 7.1 – Scénario1 : répartition des temps de séjours

Pour le test de Student, les hypothèses ont été posées comme suit :
$H_0$ : la moyenne des temps de séjour est identique entre les deux distributions
$H_1$ : la moyenne des temps de séjour est différente entre les deux distributions
Pour le scénario 1, ceci donne : $t = \frac{1888,465-1875,865}{1397,603/\sqrt{10568}} = 0.9$

Étant donné que la valeur $t*$ présente dans la table des fractiles est de 1.96, l'hypothèse $H_0$ avec une valeur $t$ de 0.9 est acceptée avec un seuil de confiance de 95%.

Ceci confirme donc notre interprétation.

**Résultats Scénario 2** Le tableau ci-dessous présente les résultats des mesures de performances des simulations effectuées avec l'Exponentielle et l'Erlang pour le scénario 2.

| Dist. | $\mu$ Tps séj. | $\sigma$ Tps séj. | % excl. | % abandon 1 | % abandon 2 | Nb cli. sys. |
|---|---|---|---|---|---|---|
| Exp. | 2476,584 | 1718,055 | 0,49 | 0,04 | 0,26 | 41 |
| Erlang | 3460,46 | 3129,03 | 0,64 | 0,04 | 0,26 | 41 |

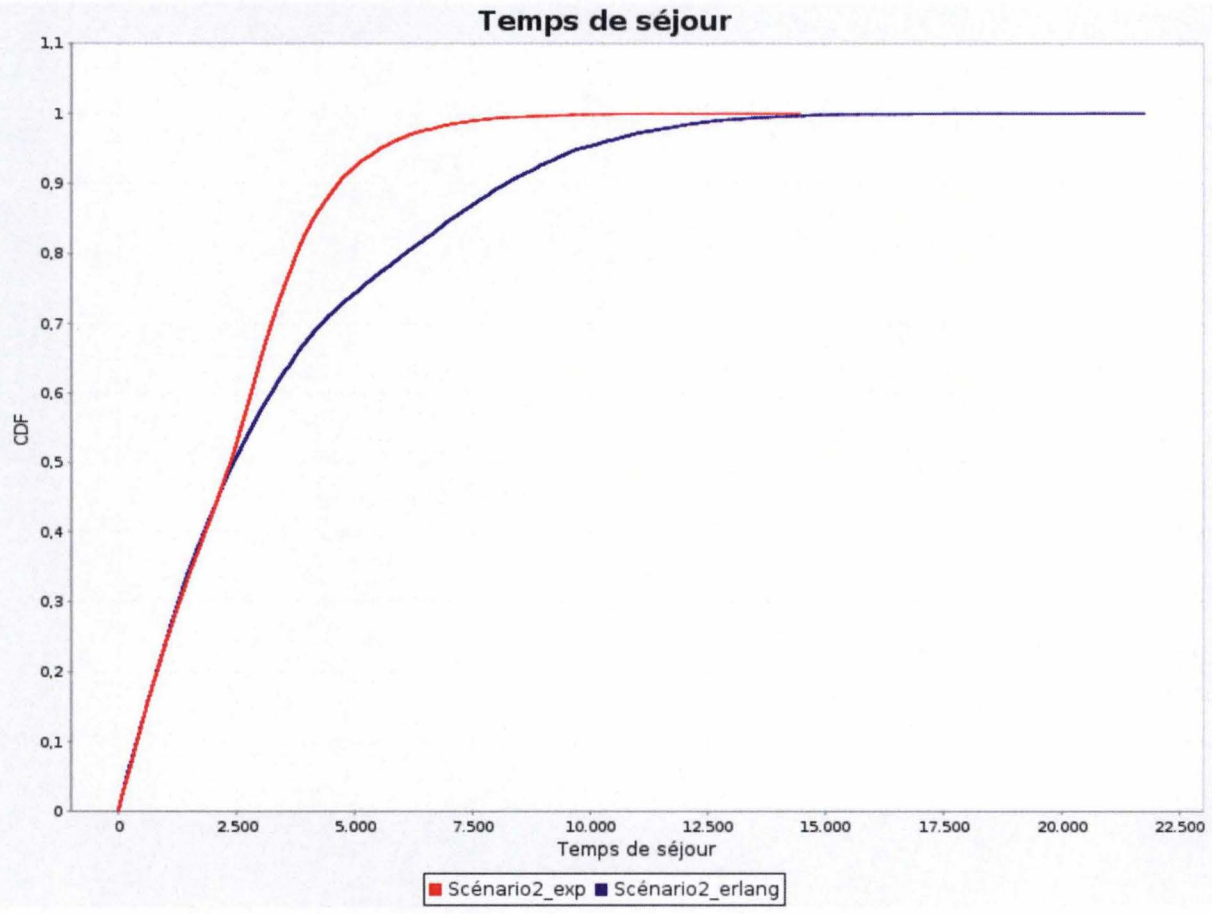



FIGURE 7.2 – Scénario2 : répartition des temps de séjours

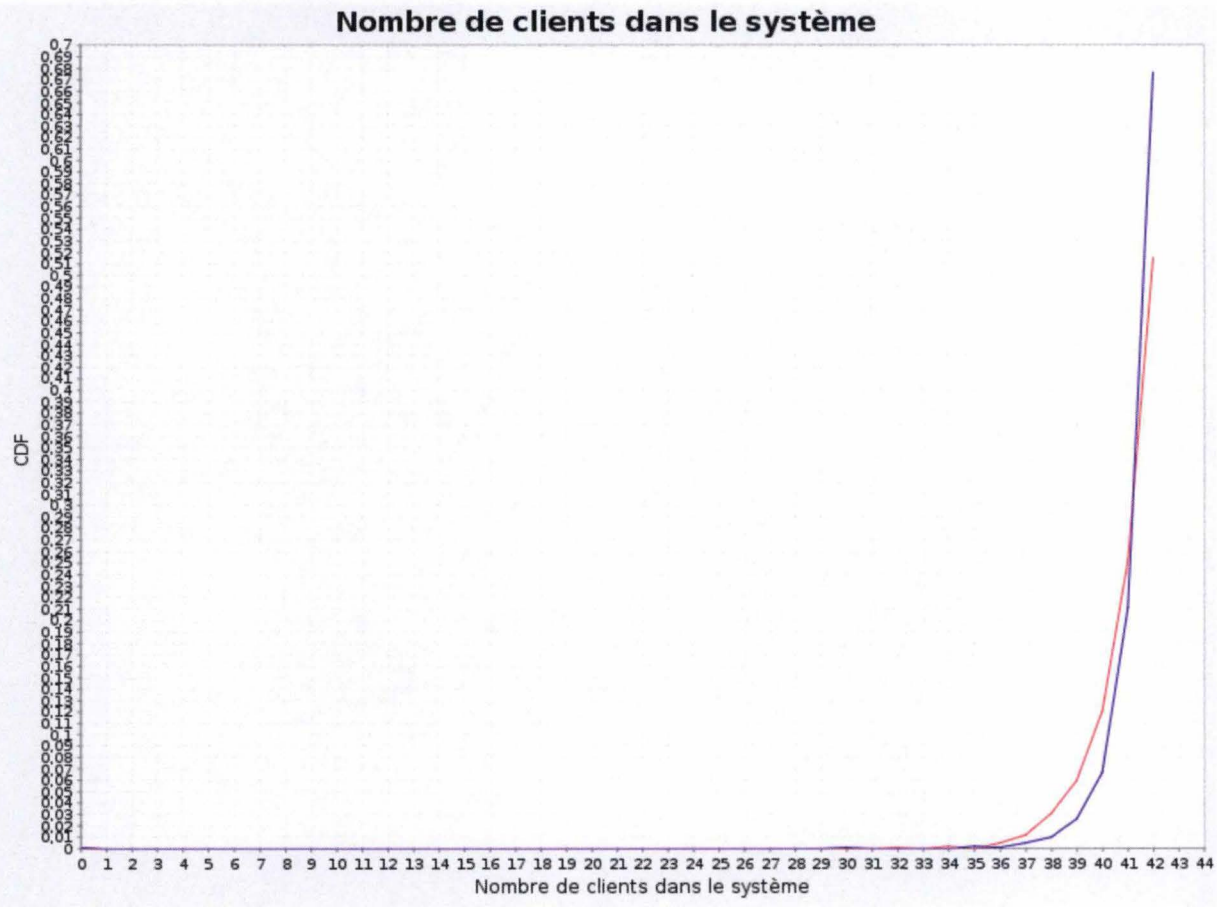



FIGURE 7.3 – Scénario2 : nombre de clients dans le système

Dans ce second scénario, nous remarquons sur le tableau ci-dessus un taux d'exclusion de clients plus faible que pour le scénario 1. Nous voyons également que la distribution Erlang possède des résultats nettement supérieur concernant le temps de séjour moyen et le taux d'exclusion. Ceci se vérifie également sur la figure 7.2 où nous remarquons des temps de séjour plus importants pour l'Erlang. Sur la figure 7.3, nous voyons également que le nombre de clients présents dans le système durant la simulation est plus élevé pour l'Erlang. Nous pouvons donc conclure pour ce scénario que les deux distributions ne sont pas identiques.

De la même manière que pour le scénario 1, nous avons réalisé le test de Student dont voici le résultat :

Pour le test de Student, les hypothèses ont été posées comme suit :
$H_0$ : la moyenne des temps de séjour est identique entre les deux distributions
$H_1$ : la moyenne des temps de séjour est différente entre les deux distributions
Pour le scénario 2, ceci donne : $t = \frac{3460,46-2746,584}{3129,03/\sqrt{11938}} = 24,92$

Étant donné que la valeur $t*$ présente dans la table des fractiles est de 1.96, l'hypothèse $H_0$ avec une valeur $t$ de 24,92 est rejetée avec un seuil de confiance de 95%.

**Résultats Scénario 3** Le tableau ci-dessous présente les résultats des mesures de performances des simulations effectuées avec l'Exponentielle et l'Erlang pour le scénario 3.

| **Dist.** | **$\mu$ Tps séj.** | **$\sigma$ Tps séj.** | **% excl.** | **% abandon 1** | **% abandon 2** | **Nb cli. sys.** |
|---|---|---|---|---|---|---|
| Exp. | 2102,616 | 1525,099 | 0,002 | 0,11 | 0,36 | 26,2 |
| Erlang | 3446,351 | 3099,414 | 0,13 | 0,10 | 0,61 | 37,7 |

Dans ce dernier scénario, nous remarquons sur le tableau ci-dessus un taux d'exclusion de clients faible. Nous voyons également que les résultats entre les deux distributions s'écartent de manière importante.

Ces observations se vérifient aussi sur les graphiques 7.4. La figure 7.4 représentant la distribution du temps de séjour montre clairement que la distribution Erlang possède des temps de séjours plus importants. Le nombre de clients présents dans le système est également plus élevé comme le montre la figure 7.5. Nous pouvons donc conclure pour ce scénario que les deux distributions ne sont pas identiques.

De la même manière que pour les deux scénarios précédents, nous avons réalisé le test de Student dont voici le résultat :

Pour le test de Student, les hypothèses ont été posées comme suit :
$H_0$ : la moyenne des temps de séjour est identique entre les deux distributions
$H_1$ : la moyenne des temps de séjour est différente entre les deux distributions
Pour le scénario 2, ceci donne : $t = \frac{3446,351-2102,616}{3099,414/\sqrt{10895}} = 45,2$

Étant donné que la valeur $t*$ présente dans la table des fractiles est de 1.96, l'hypothèse $H_0$ avec une valeur $t$ de 45.2 est rejetée avec un seuil de confiance de 95%.

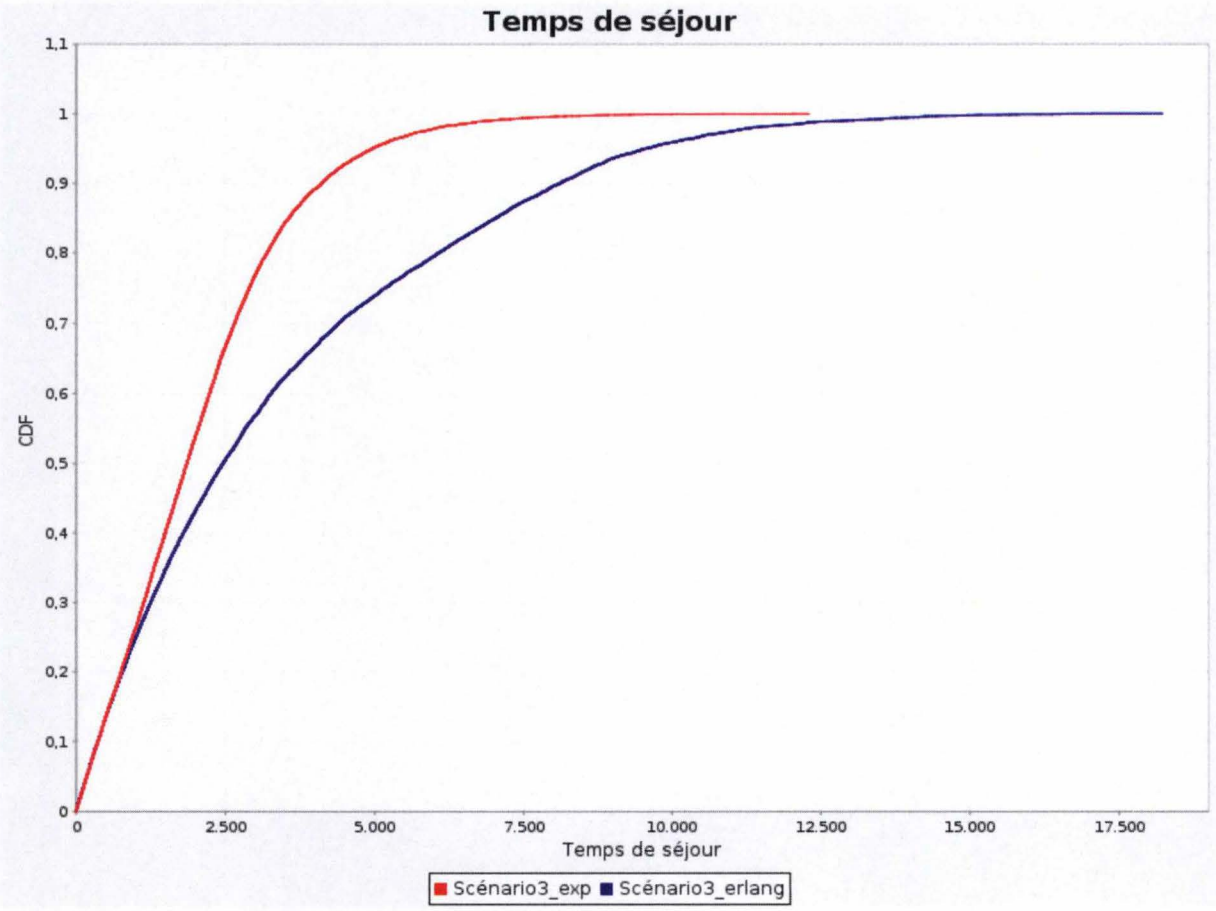



FIGURE 7.4 – Scénario 3 : répartition des temps de séjours

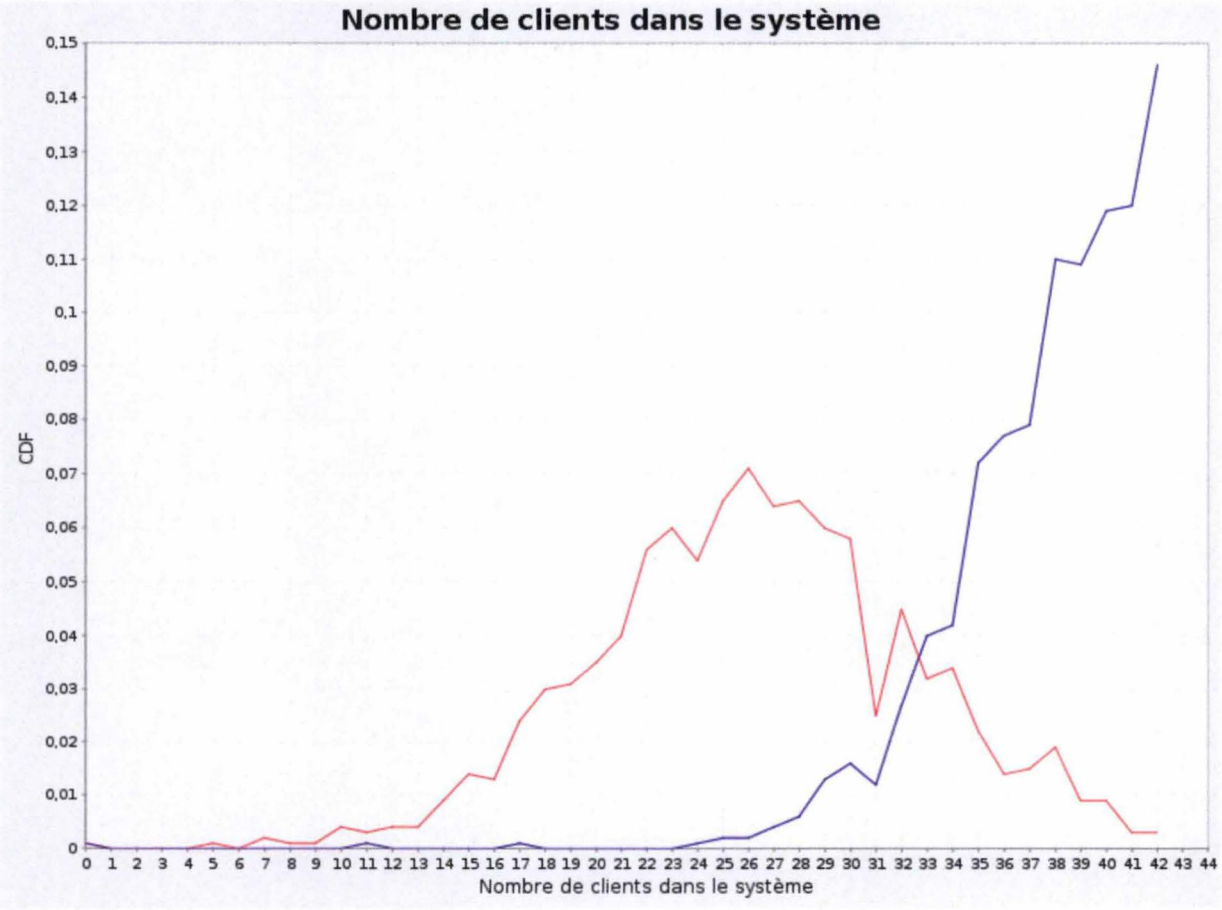



FIGURE 7.5 – Scénario 3 : nombre de clients dans le système

## 7.3 Conclusion

Dans ce chapitre, nous avons présenté les résultats de simulations effectuées avec la distribution Exponentielle et Erlang. L'objectif était confronter ces résultats afin de se rendre compte si l'hypothèse choisie au départ, à savoir l'utilisation d'une distribution exponentielle pour la taille de fichiers, présente des différences avec la distribution Erlang qui semble plus proche de la réalité. Nous pouvons dire qu'il existe un écart non négligeable entre l'Erlang et l'Exponentielle. La distribution Erlang affiche des valeurs notamment des temps de téléchargement plus importants.

# Annexe A

# Guide de l'utilisateur

## Sommaire


**A.1 Installation du simulateur** . . . . . 80
**A.2 Réaliser une simulation** . . . . . 80
**A.3 Voir les résultats d'une simulation** . . . . . 82
**A.4 Enregistrer les résultats d'une simulation** . . . . . 84
**A.5 Comparer les résultats de plusieurs simulations** . . . . . 85


Ce chapitre contient le mode d'emploi du simulateur. Les prérequis nécessaires à l'installation du simulateur sont définis. Ensuite chaque fonctionnalité du simulateur est décrite.

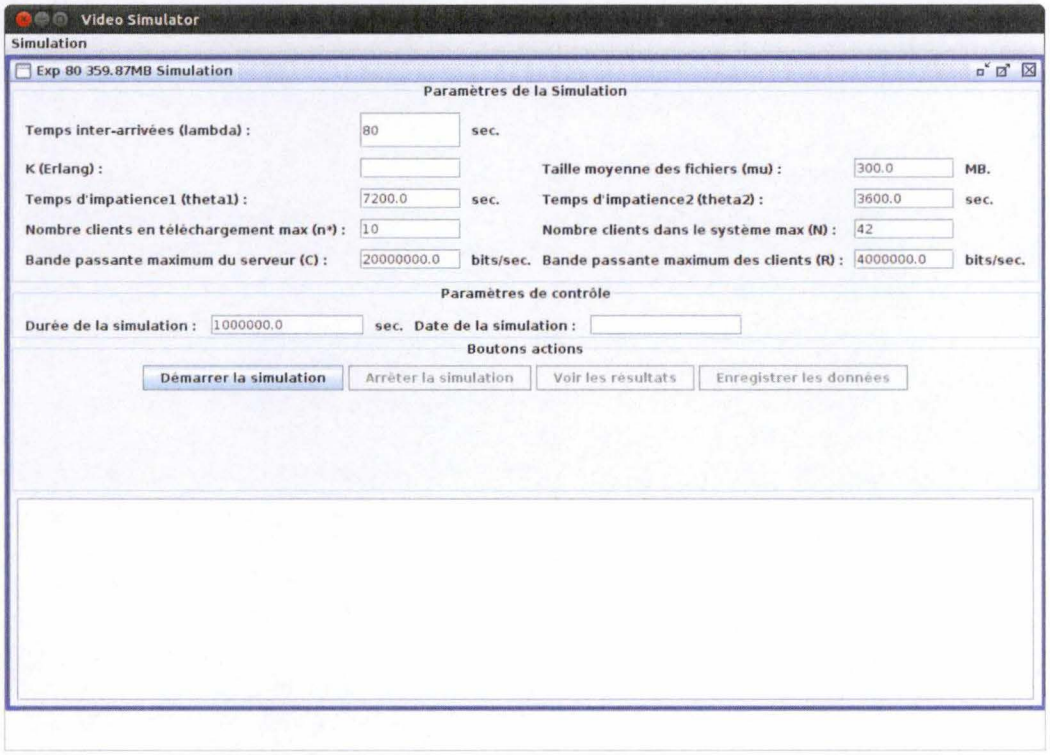
## A.1 Installation du simulateur

Le simulateur a été écrit avec le langage de programmation *Java*. Il est donc nécessaire que l'environnement *Java* soit installé au préalable. Celui-ci est disponible pour de nombreux systèmes d'exploitations et architectures. Il est téléchargeable sur le site d'Oracle [1]. Nous recommandons au minimum la version 6 de l'environnement *Java*.
Ensuite il suffit de décompresser le fichier SimVideo.zip à un emplacement présent sur le disque dur (par exemple c :\). L'application peut alors se lancer en double-cliquant sur SimVideo.jar.

## A.2 Réaliser une simulation

Pour réaliser une simulation, lancez l'application puis cliquez sur "Nouvelle simulation". Vous devez ensuite donner un nom à la simulation que vous allez effectuer. Ce nom sera utilisé notamment pour l'enregistrement de la simulation (voir A.4). L'écran suivant apparait alors devant vous :

1. `http://www.oracle.com/technetwork/java/javase/downloads/jre-6u25-download-346243.html`




Cet écran est découpé en une zone pour les paramètres d'entrée de la simulation, une zone pour les paramètres de contrôle, une zone contenant les actions disponibles du simulateur et enfin une zone console traçant le comportement du simulateur. Voici une explication des différents paramètres d'entrée du simulateur.

**Temps inter-arrivées (lambda)** Le temps des arrivées des clients est généré suivant un processus de poisson de paramètre $1/\lambda$. Le paramètre à fournir ici est $\lambda$. Il s'exprime en secondes. $\lambda$ doit être $> 0$.

**K (Erlang)** Si ce paramètre est rempli, la taille moyenne des fichiers (mu) pour les clients est générée avec une loi Erlang de paramètre $(k, 1/\mu)$. Si ce paramètre est omis la taille moyenne des fichiers (mu) pour les clients est générée avec une Exponentielle de paramètre $1/\mu$. $K$ doit être un entier positif.

**Taille moyenne des fichiers (mu)** Il s'agit de la taille moyenne des fichiers que les clients téléchargent. Cette taille sera générée soit par une Exponentielle de paramètre $1/\mu$ , soit par une Erlang de paramètre $(k, 1/\mu)$. Ce paramètre s'exprime en Megabytes. Le paramètre à fournir ici est $\mu$.

**Temps d'impatience1 (theta1)** Le temps moyen d'impatience des clients en téléchargement (theta1) est généré par une Exponentielle de paramètre $1/\theta_1$. Ce paramètre s'exprime en secondes. Le paramètre à fournir ici est $\theta_1$.

**Temps d'impatience2 (theta2)** Le temps moyen d'impatience des clients en attente (theta2) est généré par une Exponentielle de paramètre $1/\theta_2$. Ce paramètre s'exprime en secondes. Le paramètre à fournir ici est $\theta_2$.

**Nombre de client en téléchargement (n*)** Ce paramètre indique le nombre de clients que le système accepte simultanément en téléchargement. Il s'agit d'un entier. Ce nombre doit être $<$ que $N$.

**Nombre de clients dans le système (N)** Ce paramètre indique le nombre de clients total que le système accepte. Il s'agit d'un entier. Remarquez que ce $N$ = nombre de clients en téléchargement (n*) + nombre de clients en attente.

**Bande passante maximum du serveur (C)** Ce paramètre indique la bande passante maximum du serveur présent dans le système. Le serveur travaille en *Processor Sharing*, c'est donc cette bande passante qui sera partagée entre les différentes clients. Celle-ci doit être supérieure à la bande passante des clients. Ce paramètre s'exprime en bits/secondes.

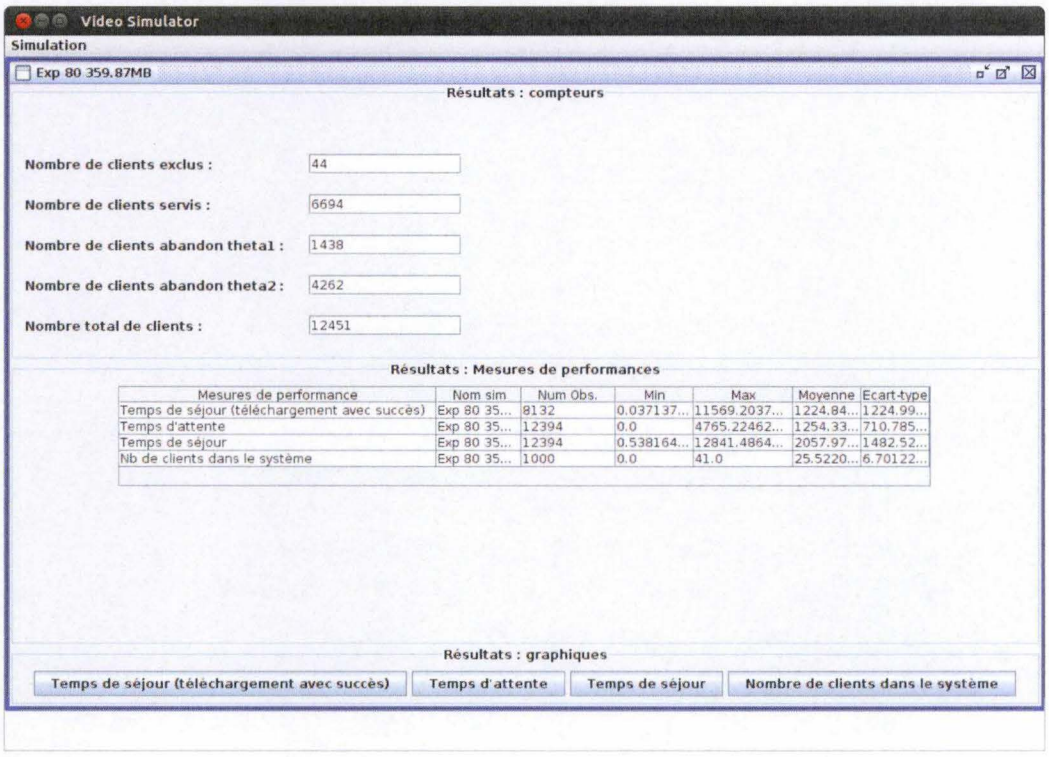
**Bande passante maximum des clients (R)** Ce paramètre indique la bande passante maximum des clients. Celle-ci doit être inférieure à la bande passante du serveur. Ce paramètre s'exprime en bits/secondes.

**Durée de la simulation** Ce paramètre indique la durée de la simulation. La simulation s'arrête automatiquement une fois ce temps écoulé. Ce temps doit être suffisamment important pour que la simulation atteigne un état stationnaire. Ce paramètre s'exprime en secondes.

**Date de la simulation** Ce paramètre est rempli par le simulateur avec la date actuelle lorsque la simulation est sauvegardée.

## A.3 Voir les résultats d'une simulation

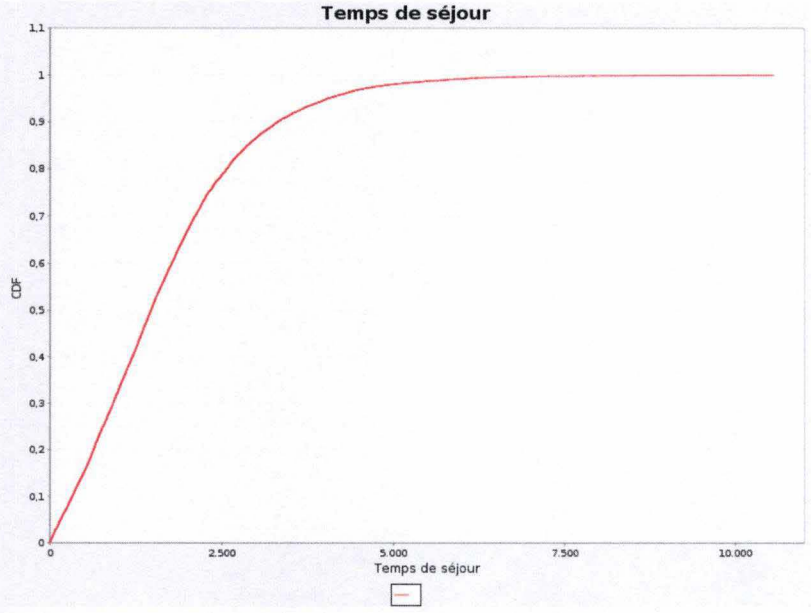
Une fois la simulation réalisée, le bouton "Voir Résultats" devient disponible. Il permet d'afficher l'écran suivant :




Cet écran est divisé en trois zones de résultats à savoir une zone "compteurs", une zone "mesures de performances" et une zone "graphiques".

La zone "compteurs" affiche le résultat de différents compteurs d'observations incrémentés lors de la simulation. La zone "mesures de performance" contient quatre mesures de performances qui ont été calculées à savoir le temps de séjour des clients ayant réussi leur téléchargement, le temps d'attente des clients, le temps de séjour des clients et le nombre de clients dans le système. Cette zone propose une représentation chiffrée de ces mesures via notamment la moyenne et l'écart-type. La zone "graphiques" propose une représentation graphique des mesures de performances citées ci-dessus. Ces graphiques représente respectivement la distribution du temps de séjour des clients ayant réussi leur téléchargement, la distribution du temps d'attente des clients, la distribution du temps de séjour des clients et la distribution du nombre de clients dans le système.

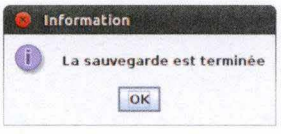
Voici un exemple de graphique pour la distribution du temps de séjour :



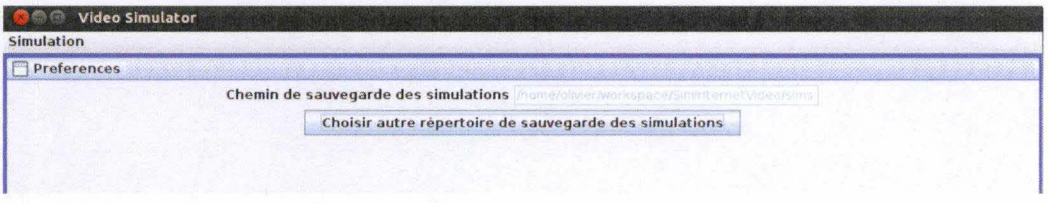

Ce graphique a été réalisé avec les paramètres par défaut proposés par l'application.

## A.4 Enregistrer les résultats d'une simulation

Une fois la simulation terminée, le bouton "Enregistrer les données" devient disponible. Celui-ci permet d'enregistrer les résultats d'une simulation dans un fichier afin de pouvoir notamment réaliser des comparaisons entre simulations. Dès que les données ont été enregistrées, la fenêtre suivante apparait :

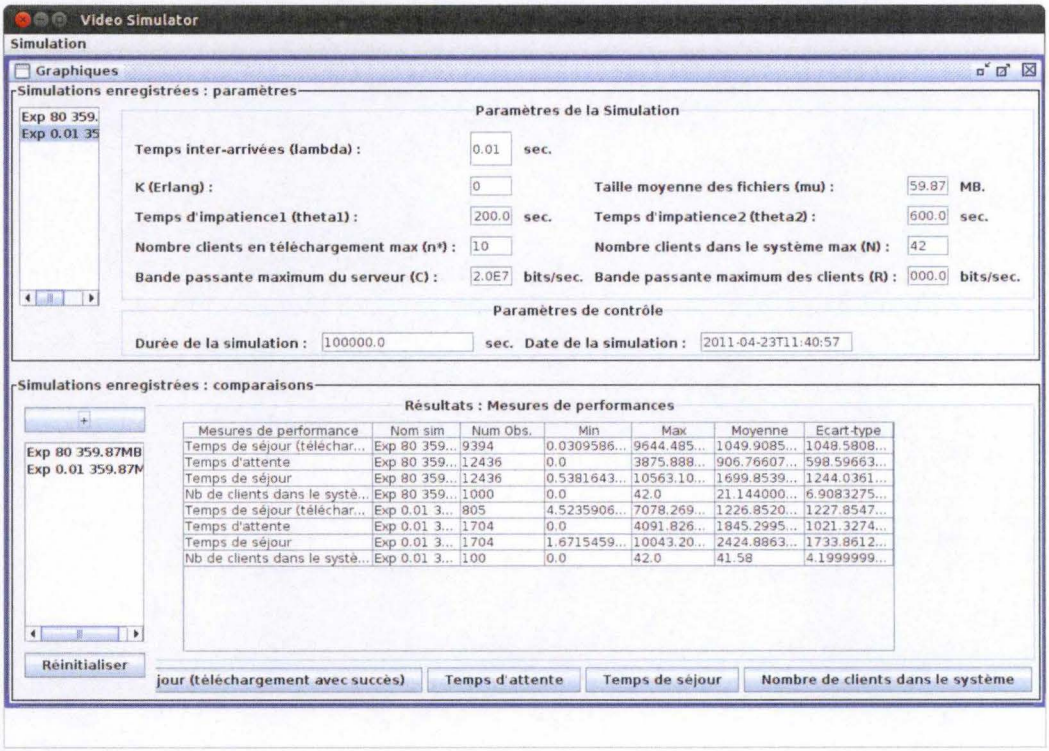



Les données sont enregistrées dans un répertoire qui est défini dans le menu "Préférences".



## A.5 Comparer les résultats de plusieurs simulations

Pour comparer les résultats de plusieurs simulations ou revoir les résultats d'une simulation, lancez l'application et cliquez sur le menu "Comparaisons". La fenêtre suivante s'ouvre alors :




Cette fenêtre est découpée en deux zones principales. La première affiche les simulations sauvegardées ainsi que les informations associées à la simulation telles que les paramètres d'entrée de la simulation. La deuxième zone permet de comparer directement plusieurs simulations notamment via des graphiques. Les jeux de données des différentes simulations choisies sont représentés sur le tableau des mesures de performances mais aussi sur les graphiques disponibles par le biais des quatre boutons en bas de l'écran.

# Annexe B

# Sauvegarde XML

Voici un exemple de fichier de sauvegarde XML comprenant les données d'une simulation.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <simulation>
3   <simulationName>oli</simulationName>
4   <simulationDate>2011-03-16T05:44:27</simulationDate>
5   <!-- parametre k de la distribution ERLANG pour le temps des
      arrivees-->
6   <param_k>0</param_k>
7   <!--paramètre lambda de la distribution Exponentielle pour le
      temps des arrivees-->
8   <param_lambda unitTime="second">0.01</param_lambda>
9   <!--la taille de fichier moyenne pour la distribution
      Exponentielle des temps de service-->
10  <param_filesize unitSize="MB">359.87</param_filesize>
11  <!--parametre theta1 de la distribution Exponentielle pour le
      temps d'impatience 1-->
12  <param_theta1 unitTime="second">300.0</param_theta1>
13  <!--parametre theta1 de la distribution Exponentielle pour le
      temps d'impatience 2-->
14  <param_theta2 unitTime="second">3600.0</param_theta2>
15  <!--nombre maximum de clients en telechargement simultanement
      -->
16  <param_nstar>10</param_nstar>
17  <!--nombre maximum de clients dans le systeme-->
18  <param_nmax>20</param_nmax>
19  <observationDatas statCollecName="Service Time Successful
      Customer">
20    <observation>36.646363499762685</observation>
21    <observation>120.83608023388616</observation>
22    ...
23  </observationDatas>
24  <observationDatas statCollecName="Waiting Times">
25    <observation>0.0</observation>
```

```
26      <observation>0.0</observation>
27      <observation>36.60115609616254</observation>
28      ...
29    </observationDatas>
30    <observationDatas statCollecName="Sejour Times">
31      <observation>36.646363499762685</observation>
32      <observation>120.83608023388616</observation>
33      <observation>126.20298197945607</observation>
34      ...
35    </observationDatas>
36    <nbCustomerExclusion>50015</nbCustomerExclusion>
37    <nbCustomerServed>3</nbCustomerServed>
38    <nbCustomerLeaveTheta1>11</nbCustomerLeaveTheta1>
39    <nbCustomerLeaveTheta2>0</nbCustomerLeaveTheta2>
40    <nbCustomerAll>50049</nbCustomerAll>
41 </simulation>
```

Voici le schéma de validation XML qui a été utilisé. Celui-ci a pour objectif la vérification syntaxique du fichier XML.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3
4  <!-- Définition d'un type unité de temps -->
5  <xsd:simpleType name="unitTime">
6    <xsd:restriction base="xsd:string">
7      <xsd:enumeration value="second"/>
8      <xsd:enumeration value="minute"/>
9      <xsd:enumeration value="hour"/>
10   </xsd:restriction>
11 </xsd:simpleType>
12
13 <!-- Définition d'un type unité de taille de fichier
       informatique -->
14 <xsd:simpleType name="unitSize">
15   <xsd:restriction base="xsd:string">
16     <xsd:enumeration value="KB"/>
17     <xsd:enumeration value="MB"/>
18     <xsd:enumeration value="GB"/>
19   </xsd:restriction>
20 </xsd:simpleType>
21
22 <!-- Début schéma -->
23 <xsd:element name="simulation">
24   <xsd:complexType>
```

```
25  <xsd:sequence>
26          <xsd:element name="simulationName" minOccurs="1"
              maxOccurs="1">
27      <xsd:simpleType>
28      <xsd:restriction base="xsd:string">
29              <xsd:minLength value="1"/>
30                  <xsd:maxLength value="255"/>
31          </xsd:restriction>
32      </xsd:simpleType>
33        </xsd:element>
34      <xsd:element name="simulationDate" minOccurs="1" maxOccurs
          ="1">
35      <xsd:simpleType>
36          <xsd:restriction base="xsd:dateTime"/>
37        </xsd:simpleType>
38    </xsd:element>
39
40    <xsd:element name="param_k" minOccurs="1" maxOccurs="1" type
        ="xsd:integer"/>
41
42    <xsd:element name="param_lambda" minOccurs="1" maxOccurs="1"
        >
43      <xsd:complexType>
44      <xsd:simpleContent>
45            <xsd:extension base="xsd:decimal">
46            <xsd:attribute name="unitTime" type="unitTime" use="
                required"/>
47            </xsd:extension>
48          </xsd:simpleContent>
49      </xsd:complexType>
50    </xsd:element>
51
52    <xsd:element name="param_filesize" minOccurs="1" maxOccurs="
        1">
53      <xsd:complexType>
54      <xsd:simpleContent>
55            <xsd:extension base="xsd:decimal">
56            <xsd:attribute name="unitSize" type="unitSize" use="
                required"/>
57            </xsd:extension>
58          </xsd:simpleContent>
59      </xsd:complexType>
60    </xsd:element>
61
62    <xsd:element name="param_theta1" minOccurs="1" maxOccurs="1"
        >
```

```
63        <xsd:complexType>
64        <xsd:simpleContent>
65              <xsd:extension base="xsd:decimal">
66              <xsd:attribute name="unitTime" type="unitTime" use="
                   required"/>
67              </xsd:extension>
68            </xsd:simpleContent>
69        </xsd:complexType>
70      </xsd:element>
71
72      <xsd:element name="param_theta2" minOccurs="1" maxOccurs="1"
           >
73        <xsd:complexType>
74        <xsd:simpleContent>
75              <xsd:extension base="xsd:decimal">
76              <xsd:attribute name="unitTime" type="unitTime" use="
                   required"/>
77              </xsd:extension>
78            </xsd:simpleContent>
79        </xsd:complexType>
80      </xsd:element>
81
82
83      <xsd:element name="param_nstar" minOccurs="1" maxOccurs="1"
           type="xsd:integer"/>
84
85      <xsd:element name="param_nmax" minOccurs="1" maxOccurs="1"
           type="xsd:integer"/>
86
87          <xsd:element name ="observationDatas" maxOccurs="
               unbounded">
88              <xsd:complexType>
89                  <xsd:sequence>
90                      <xsd:element name="observation" type="
                           xsd:decimal" maxOccurs="unbounded"/>
91                  </xsd:sequence>
92        <xsd:attribute name="statCollecName" use="required"/>
93                  </xsd:complexType>
94              </xsd:element>
95
96      <xsd:element name="nbCustomerExclusion" minOccurs="1"
           maxOccurs="1" type="xsd:decimal"/>
97      <xsd:element name="nbCustomerServed" minOccurs="1" maxOccurs
           ="1" type="xsd:decimal"/>
98      <xsd:element name="nbCustomerLeaveTheta1" minOccurs="1"
           maxOccurs="1" type="xsd:decimal"/>
```

```
99      <xsd:element name="nbCustomerLeaveTheta2" minOccurs="1"
          maxOccurs="1" type="xsd:decimal"/>
100     <xsd:element name="nbCustomerAll" minOccurs="1" maxOccurs="1
          " type="xsd:decimal"/>
101
102   </xsd:sequence>
103   </xsd:complexType>
104 </xsd:element>
105 </xsd:schema>
```

# Annexe C

# Données test

Nous pouvons donner au simulateur un jeu de données "test" afin de vérifier son comportement. Cette fonctionnalité n'est pas disponible via l'interface graphique. Pour tester le simulateur, un fichier CSV du nom de "InputDatasTest.csv" doit être placé dans le répertoire InputDatas présent à la racine du répertoire du simulateur. Ci-dessous un exemple de ce fichier ainsi qu'une explication sont donnés. Comme nous le voyons sur le tableau ci-dessous, chaque ligne définit les donnes d'entrée pour un client.

| delayTime | fileSize | bandwidth | num client | impatience1 | impatience2 |
|---|---|---|---|---|---|
| 60 | 838860800 | 4000000 | 1 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 2 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 3 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 4 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 5 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 6 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 7 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 8 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 9 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 10 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 11 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 12 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 13 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 14 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 15 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 16 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 17 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 18 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 19 | 300 | 3600 |
| 60 | 838860800 | 4000000 | 20 | 300 | 3600 |

Voici une explication des différentes colonnes :

- La colonne *delayTime* indique l'instant d'arrivée du client de manière relative, c'est-à-dire par rapport au client précédent. Ce temps est exprimé en secondes.

- La colonne *fileSize* indique la taille de fichier que le client téléchargera, exprimé en bits.
- La colonne *bandwidth* indique la bande passante de téléchargement du client, exprimé en bits/s.
- La colonne *num client* permet de définir un numéro pour le client. Ceci permet de tracer plus facilement un client par la suite.
- La colonne *impatience1* indique le temps d'impatience en téléchargement pour ce client. Ce temps est exprimé en secondes.
- La colonne *impatience2* indique le temps d'impatience en attente pour ce client. Ce temps est exprimé en secondes.

# Bibliographie


[1] R. E. Tarjan D. D. Sleator. Self-adjusting binary search trees. 1985.

[2] Keith W. Ross James F. Kurose. *Computer Networking : a top-down approach.* Addison Wesley, 4nd edition, 2008. ISBN 978-0-321-49770-3.

[3] Stephen K.Park Lawrence M. Leemis. *Discret-Event Simulation : A First Course.* Pearson Education, 2006. ISBN 0-13-142917-5.

[4] Nicolas Menneceur. Quelques mots sur la technologie du streaming. 2006.

[5] Marie-Ange Remiche Tobias Hobfeld, Kenji Leibnitz. Exact sejour time distribution in an online iptv recording system. 2008.

[6] Huseyin Uzunalioglu. Channel change delay in iptv systems.

[7] Howard Greenfield Wes Simpson. *IPTV and Internet Video : Expanding the Reach of Television Broadcasting* . Elsevier Science Technology, United Kingdom, 2nd edition, 2009. ISBN 978-0-240-81245-8.

[8] Dony Yves. Video-on-demand over internet : a survey of existing systems and solutions. Master's thesis, University of Namur (FUNDP), Belgium.