République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira-Bejaïa
Faculté des Sciences Exactes
Département d'informatique

Mémoire de fin du cycle

En vue de l'obtention du Diplôme de Master professionnel en Informatique

Option : Administration et sécurité des réseaux

Thème

Conception et réalisation d'un module de gestion de données pour PeerSim

Réalisé par :

BENAISSA Salah

BOUAOUINA Amirouche

Encadré par :

M. SEBAA Abderezzak

Membres de jury :

Président : Mme BOUTRID Samia

Examinateur : M. DJEBARI Nabil

Promotion : 2013/2014

Remerciements

En premier lieu, nous remercions DIEU le tous puissant pour son aide, de nous avoir donné la volonté, le courage et la patience pour accomplir ce travail.

Notre gratitude et profonde reconnaissance s'adressent à notre promoteur Mr SEBAA Abderezzak. Pour avoir accepté de nous encadrer ainsi que pour la confiance et le grand soutien, pour ses remarques bienveillantes et son orientation, pour ses conseils précieux qu'il nous a prodigué tout le long de notre travail.

Aux membres du jury pour l'honneur qu'ils nous ont accordé en jugeant ce travail, notamment :

A Mme BOUTRID Samia d'avoir accepté de présider le jury.

A M. DJEBARI Nabil qui nous a fait l'honneur d'examiner ce travail.

Nos vifs remerciements sont adressés plus particulièrement à nos familles et nos amis(es) qui ont su nous soutenir, nous encourager, nous aider tout au long des années.

Un grand merci à tous ceux qui ont contribué de prés ou de loin à l'élaboration de ce modeste travail, ils trouvent ici l'expression de nos sincères reconnaissances.

Dédicaces

Je dédie ce modeste travail :

A la mémoire de mes chers grands parents paternels et maternels qui sont dignes de ma gratitude et mon estime.

A mes très chères parents qui m'ont toujours soutenu pendant toutes mes études, en témoignage d'affection et de profondes reconnaissances envers eux, et aux quels je rendrai assez.

A toute ma famille qui ont su m'encourager, m'aider et me supporter tout au long des années.

A tous mes amis.

A tous ceux qui me sont chers.

Amirouche

Dédicaces

Je dédie ce modeste travail :

A la mémoire de mes chers grands parents paternels et maternels qui sont dignes de ma gratitude et mon estime.

A mes très chères parents qui m'ont toujours soutenu pendant toutes mes études, en témoignage d'affection et de profondes reconnaissances envers eux, et aux quels je rendrai assez.

A toute ma famille qui ont su m'encourager, m'aider et me supporter tout au long des années.

A tous mes amis.

A tous ceux qui me sont chers.

Salah

TABLE DES MATIÈRES

Chapitre III : ***Gestion de données dans les réseaux P2P***

Chapitre IV: ***Analyse et conception***

Chapitre V: *Réalisation*

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS

SBD: système de bases de données.

SGBD: système de gestion de Base de données.

BDD : base de données.

P2P: peer-to-peer (pair-à-pair).

XML: eXtensible Markup Language.

CPU: Central Processing Unit.

UML: Unified Modeling Language.

2TUP: 2 tracks unified process.

API: Application Programming Interface.

SWT: Standard Widget Toolkit.

IBM: International Business Machines.

JVM: Java Virtual Machine.

Introduction générale

Au cours des dernières années, Les technologies Peer-to-Peer(P2P) sont devenues de plus en plus populaire. Les Systèmes de P2P peuvent être définis comme une architecture de réseau distribué, où les participants partagent une partie de leurs propres ressources matérielles, telles que la puissance de traitement, capacité de stockage, de bande passante réseau. Les ressources partagées sont nécessaires pour fournir le service et le contenu offert par le réseau, comme le partage de fichiers, la gestion de données. Le service ou contenu fourni par le réseau P2P est accessible par d'autres pairs directement, sans passer par des entités intermédiaires.

Le développement de système de P2P a beaucoup de problèmes communs trouvés avec la programmation d'autres systèmes répartis. Les systèmes de P2P ajoutent d'autres complications, particulièrement en examinant les différentes topologies et en installant des scénarios à travers les machines multiples. Le temps pris pour développer un système de P2P peut être plus long, en raison de ces problèmes. Les simulateurs de réseau de P2P essayent de résoudre la difficulté accrue en concevant ou en examinant des systèmes de P2P, en fournissant un environnement virtuel de réseau.

Les simulateurs de réseau de P2P sont généralement utilisés pour simuler les communications réseau dans des scénarios ou situations particulières, sans avoir à configurer les machines ou les réseaux réels. Ils peuvent aider au développement et à l'essai d'une application réseau. Les simulateurs de réseau de P2P permettent aux développeurs de produire une topologie de réseau et définir la largeur de bande et les caractéristiques de connexion et de circulation pour les nœuds et les liens.

Ils existent plusieurs simulateurs ; **PeerSim** est l'un des plus connue et fait l'objet de notre projet. Il a l'avantage d'être déjà spécialisé pour l'étude des systèmes P2P et qui dispose d'une architecture ouverte, et une approche modulaire qui permet de l'adapter et de le spécifier. **PeerSim** propose un grand nombre de ses modules dans ses sources, qui facilitent grandement le codage de nouvelles applications.

L'objectif visé par ce projet, est de concevoir et implémenter un module de gestion de données pour **PeerSim**, qui permet de gérer les différents types de données qui circulent dans les réseaux P2P, et pouvoir créer et initialiser des requêtes à travers le réseau P2P. Le module

doit permettre de gérer les données de telle sorte que l'utilisateur peut ajouter ou supprimer une donnée, ainsi que la gestion des requêtes. Pour aboutir à cet objectif, notre travail est organisé en cinq chapitres :

Le premier chapitre appelé « ***Généralité sur P2P*** » a pour but de définir les concepts essentiels sur lesquels est fondé notre travail.

Le deuxième chapitre appelé « ***Généralité sur PeerSim*** » a pour but de faire une étude sur le simulateur **PeerSim**

Le troisième chapitre intitulé « ***Gestion de données dans les réseaux P2P*** » est consacré pour définir les concepts liés à la de gestion de données dans les réseaux P2P, ainsi que les différents types de données qui circulent dans les réseaux P2P.

Le quatrième chapitre vise « ***l'analyse et la conception*** » qui est le cœur de notre projet. Dans lequel nous allons présenter la problématique et l'objectif de notre projet, nous allons recenser les principaux utilisateurs de notre application, ensuite nous présenterons les diagrammes de séquences dont l'objectif est de représenter les interactions entre les objets de notre système, finalisons par le diagramme de classes qui donne une vision générale et simple sur le système avant l'implémentation.

Le dernier chapitre est consacré à « ***La réalisation*** », nous définissons les outils de développement que nous avons utilisé, nous présentons également quelques captures du projet développé.

Nous achevons ce travail avec une conclusion générale résumant les connaissances acquises durant la réalisation du projet.

Chapitre *I*

Généralités sur P2P

« Savoir travailler, tout est là. »

Marmontel

Introduction

Avec l'arrivée de l'Internet et son rôle actif croissant dans l'économie et la société, l'informatique réseau n'a eu de cesse de trouver des innovations pour exploiter les ressources qu'un réseau de cette ampleur contient. Le monde de l'informatique est en effervescence autour d'un phénomène portant le nom de peer-to-peer. Mal identifiée, mal comprise et mal considérée à sa naissance, l'idée a beaucoup mûrie aux cours des dernières années. Aujourd'hui, on parle sérieusement du peer-to-peer comme un modèle de communication capable de changer radicalement certaines approches de l'informatique en réseau.

Dans ce chapitre nous allons voir quelques généralités sur P2P dont on aura besoin dans la suite de se travail.

1. Généralité sur P2P

1.1. Définition

Le terme "Peer-to-Peer", "poste à poste" ou "pair à pair" en français (ou plus couramment P2P), désigne un modèle de réseau informatique dont les éléments (les nœuds ou "peer") sont à la fois clients et serveurs lors des échanges. Ce modèle a permit une décentralisation des réseaux, en offrant une alternative aux traditionnels architectures client/serveur.

Comme leur nom l'indique, les Peer-to-Peer permettent d'établir des communications directes, d'égal à égal, entre les différents nœuds du réseau, qui peuvent alors échanger différents types d'informations sans passer par un serveur central **[1].**

1.2. Caractéristiques des réseaux P2P

- Localisation des fichiers dans un environnement distribué.
- Libre circulation des fichiers entre les pairs.
- Capacité de connexion variable suivant les pairs.
- Hétérogénéité du réseau suivant les pairs.
- Echanges d'information non sécurisés.
- Certains pairs peuvent être non fiables.
- Aucune vue globale du système.

1.3. Comparaison avec le modèle client/serveur

Le tableau (**Tableau 1**) ci-dessous résume les différences des architectures Client/serveur et Peer-To-Peer **[2].**

P2P	Client/serveur
Auto-organisé	Management centralisé
Evolution dynamique	Configuration statique
Découverte des pairs	Consultation de tables
Flux distribué	Flux centralisé
Symétrie du réseau	Asymétrie du réseau
Adressage dynamique (appli.)	Adressage statique (IP)
Nœuds autonomes	Nœuds dépendants
Attaques difficiles	Attaques plus simples

Tableau I.1 : Client/serveur vs Peer-To-Peer.

1.4. Architecture peer to peer

1.4.1. Architecture centralisée

Dans toute architecture centralisée, un dispositif exclusivement serveur se charge de mettre en relation directe tous les utilisateurs connectés. L'intérêt de cette technique réside dans l'indexation centralisée de tous les répertoires et intitulés de fichiers partagés par les abonnés sur le réseau. En général, la mise à jour de cette base s'effectue en temps réel, dès qu'un nouvel utilisateur se connecte ou quitte le service.

Cela fonctionne avec le client comme avec un moteur de recherche classique : on lance une requête en inscrivant un mot clé. On obtiendra une liste d'utilisateurs actuellement connectés au service et dont les fichiers partagés correspondent au terme recherché. Dès lors, il suffit de cliquer sur un des intitulés de lien pour se connecter directement sur la machine correspondante et entamer le transfert. Dans ces conditions, à aucun moment les fichiers se retrouvent stockés sur le serveur central **[3]**.

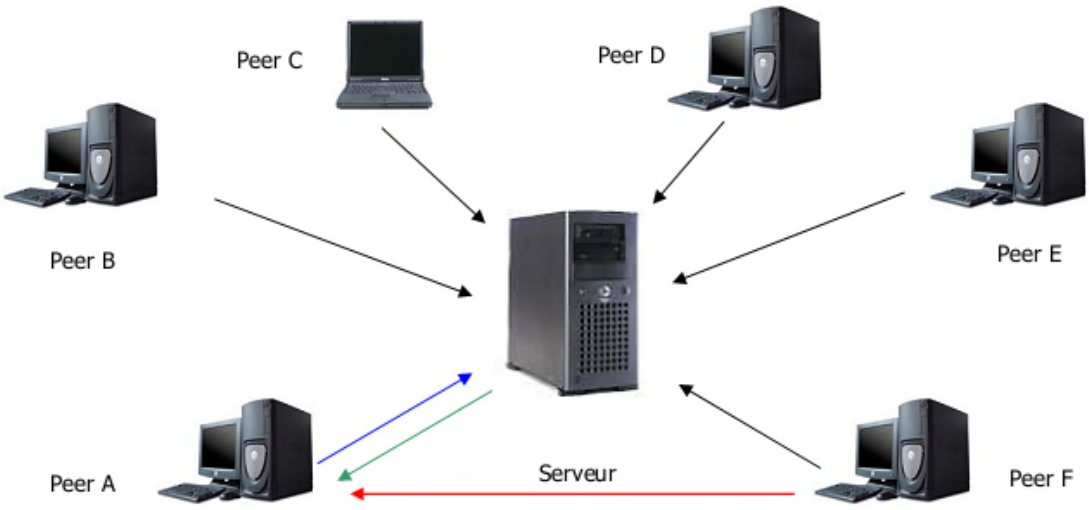

Figure I.1 : schéma d'une architecture centralisée.

- Un utilisateur recherche un fichier ressource en envoyant une requête au serveur central.
- Le serveur central répond et transmet la liste des ordinateurs utilisateurs proposant le fichier demandé.
- L'utilisateur télécharge le fichier directement à partir d'un des ordinateurs renseignés par le serveur.

1.4.2. Architecture décentralisée

Contrairement aux réseaux centralisés, où il suffisait de se connecter au serveur pour avoir accès aux informations, il faut pour avoir accès à une information en décentralisé :

- Apprendre la topologie du réseau sur lequel le client est connecté.
- Rechercher l'information sur plusieurs nœuds.
- Recevoir une réponse d'un nœud répondant aux critères.

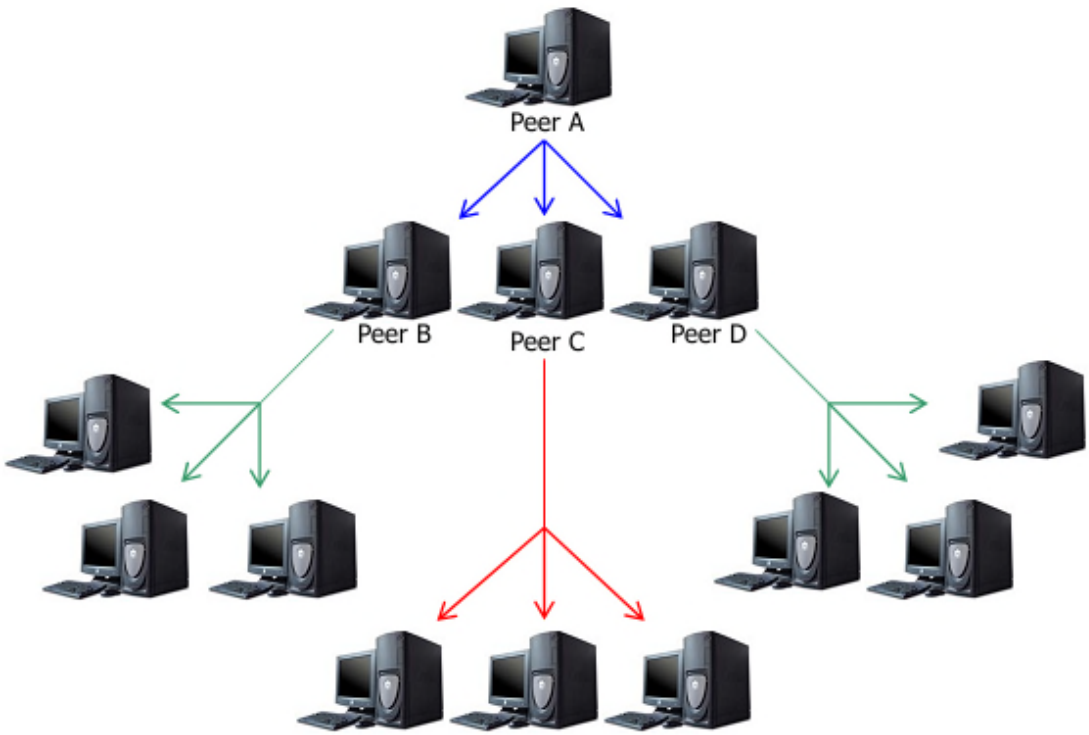

Figure I.2 : schéma d'une architecture décentralisée.

- Le client A se connecte sur le réseau, il ne connaît pas la topologie du réseau (A est totalement aveugle).
- Pour connaître les autres membres du réseau, A va "broadcaster" une demande d'identification des nœuds du réseau.
- Les nœuds recevant la demande vont à leur tour la répercuter sur tous les nœuds voisins et ainsi de suite (*comme les nœuds B, C et D*).
- Lorsque la trame est reçue et identifiée par un autre client, le nœud renvoi une trame d'identification à A.
- Ainsi, A va peu à peu pouvoir identifier tous les nœuds du réseau et se créer un annuaire **[3]**.

1.4.3. Architecture hybride

Le réseau hybride est le plus complexe à mettre en œuvre et combine à la fois le réseau centralisé et distribué. Le réseau de ce type s'appuie sur un ensemble de serveurs (l'ordinateur d'un utilisateur peut devenir un serveur) gérant un groupe d'utilisateurs suivant l'architecture centralisée. Puis, chaque serveur est ensuite connecté à d'autres serveurs suivant l'architecture

distribuée. De cette façon, si un fichier recherché par un utilisateur n'est pas indexé par le serveur auquel il est rattaché, celui-ci transmet alors la requête à un autre serveur. Cette architecture permet de bénéficier d'une meilleure bande passante en réduisant le trafic de requêtes **[4].**

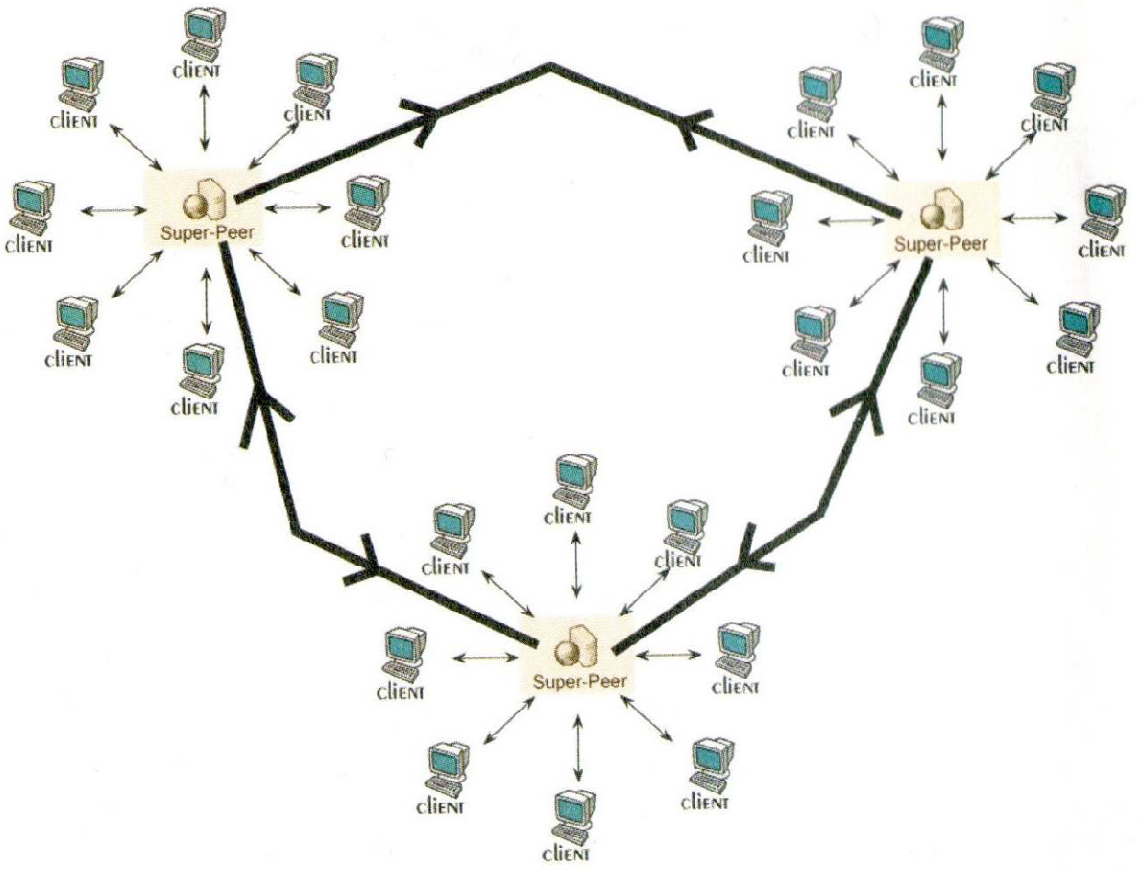

Figure I.3: schéma d'une architecture hybride.

1.5. Les avantages et les inconvénients

1.5.1. Avantages

Les architectures P2P présentent beaucoup d'avantages **[5]** :

- **Répartition de la charge** : Contrairement aux réseaux client/serveur ou le serveur principal est chargé de gérer l'ensemble des activités du réseau, d'où un risque de saturation, dans un réseau P2P, ce sont tous les pairs qui contribuent à la gestion des différentes requêtes et des ressources disponibles.
- **Capacité de stockage découplée** : Il est a savoir que dans un réseau à serveur central, toutes les données sont stockées au niveau de ce serveur, ce qui vient en opposition aux réseaux P2P ou bien que chaque pair n'offre qu'un petit espace disque

(relativement au serveur) la capacité de stockage au sein du réseau est décuplée grâce a la contribution de tous les pairs.

- **Puissance de calcul** : Des études **[6]** avancent que la puissance de calcul utilisée en moyenne par un utilisateur est de 20%, le processeur est donc sous exploite. Certaines applications P2P développées dans le domaine de la recherche (Seti@home, Décryptons…) visent à se servir de cette puissance inutilisée pour réaliser des taches reparties qu'un simple ordinateur ne pourrait accomplir en un temps raisonnable.
- **Réseaux très extensibles :** Une particularité des réseaux P2P est qu'ils sont de nature ≪ Ad-hoc ≫ c'est-a-dire que des nœuds peuvent apparaitre et disparaitre à tout moment, cette gestion dynamique des pairs facilite l'intégration de nouveaux pairs au sein du réseau.
- **Résistance aux pannes :** Dans un réseau P2P, les ressources sont reparties sur l'ensemble des utilisateurs connectes, la panne d'un pair ne peut donc pas altérer le fonctionnement du réseau.
- **Disponibilité accrue des ressources :** Comme les réseaux P2P sont extensibles, et que les pairs sont fournisseurs de ressources au sein du réseau, plus le nombre d'utilisateurs augmente, plus la disponibilité des ressources présentes augmente.
- **Diversité des chemins dans le réseau** : La topologie logique du réseau P2P étant maillée, plusieurs chemins de communication sont possibles entre chaque couple de pairs. De plus, les échanges se font plus rapidement étant plus directs.
- **Maintenance et coûts réduits** : Le serveur d'une architecture client/serveur reste très gourmand en matière de ressources, sa maintenance est très fastidieuse et son cout peut s'avérer très élève. De ce fait, l'absence de celui-ci dans les architectures P2P rend ces réseaux peu couteux.
- **Anonymat** : Certains algorithmes de routage ne permettent pas le pistage d'une requête, garantissant ainsi l'anonymat aux utilisateurs.
- **Meilleure exploitation de la bande passante** : Dans les réseaux a serveurs centraux, les goulots d'étranglements sont relativement fréquents au niveau de ces derniers, ce qui paralyse le réseau, ainsi l'absence d'organisme central dans les réseaux P2P facilite la circulation des flux, et augmente par conséquent l'utilisation de la bande passante.

1.5.2. Inconvénients

Les réseaux P2P rencontrent de nombreux problèmes et présentent quelques inconvénients, on peut citer essentiellement **[6]** :

- **Topologie instable** : Les pairs d'un réseau P2P étant dynamiques, ils peuvent apparaitre et disparaitre a tout moment, par conséquent les ressources aussi.
- **Sécurité** : Nous pouvons citer quelques exemples : Crackers, Virus, Distributed Deny of Service/DDoS, Confidentialité, Authentification **[6].**
- **Contenu trompeur** : Un des inconvénients majeurs des applications P2P est que les données circulent librement dans le système, sans vérification. Certains fichiers peuvent être corrompus : mauvaise qualité, défectueux, contenu non correspondant a l'intitule.
- **QoS (Quality of Service)** : Bien que très utilisées dans d'autres domaines, beaucoup d'applications P2P sont employées de nos jours pour le téléchargement souvent illégal de fichiers, ce qui pousse certains fournisseurs à vouloir limiter au maximum les flux P2P au sein du réseau. La bande passante allouée aux applications P2P est donc considérablement réduite, rendant ainsi les échanges P2P très lents.
- **Loi du Wild Wild Web**: Les applications P2P sont devenues la cible de plusieurs organismes de protection des droits d'auteurs et lutte contre les contenus immoraux.
- **Régulation/Répression** : Certains modèles de réseaux P2P garantissent l'anonymat aux utilisateurs, il est donc plus difficile aux autorités d'appliquer les lois.

Conclusion

Dans ce chapitre, nous avons souhaité exposer les réseaux pair-à-pair et mettre en avant le fait qu'il y ait existe différents modèles, chacun disposant d'avantages et d'inconvénients. De nombreux travaux sont effectués récemment sur le paradigme pair-à-pair permettant ainsi l'apparition de plusieurs applications.

Dans le chapitre suivant nous allons faire une étude sur le simulateur des réseaux pair à pair PeerSim.

Chapitre *II*

Généralités sur PeerSim

« Savoir travailler, tout est là. »

Marmontel

Introduction

La simulation des réseaux Pair-à-Pair (P2P) est un problème commun pour des chercheurs et des développeurs. Pour cela il a fallut créer des simulateurs qui répond a leurs exigences. Ils existent plusieurs simulateurs, **PeerSim** est l'un des plus connu et fait l'objet de notre projet. Il a l'avantage d'être déjà spécialisé pour l'étude des systèmes P2P et qui dispose d'une architecture ouverte, et une approche modulaire qui permet de l'adapter et de le spécialiser. De plus, il est soutenu par une équipe de développement active, ce qui permet d'avoir à notre disposition un nombre de modules de base déjà implémentés. Ces modules peuvent être de différents types, par exemple il y a des modules qui peuvent construire et initialiser le réseau, les modules qui peuvent gérer les différents protocoles, les modules pour contrôler et modifier le réseau. **PeerSim** propose un grand nombre de ces modules dans ses sources, qui facilitent grandement le codage de nouvelles applications.

Dans ce chapitre nous allons voir quelques généralités sur le simulateur **PeerSim**.

1. Présentation de PeerSim

Le simulateur **PeerSim** est un ensemble de classes java qui implémentent un simulateur de réseau pair à pair. Il a été conçu pour être dynamique et extensible. Sa philosophie consiste à employer une approche modulaire, car le moyen privilégié du codage avec lui est de réutiliser les modules existants. Ces modules peuvent être de différentes sortes, par exemple il y a des modules qui peuvent construire et initialisent le réseau, des modules qui peuvent manipuler les différents protocoles, modules pour commander et modifier le réseau. PeerSim offre beaucoup de ces modules dans ses sources, qui soulagent considérablement le codage de nouvelles applications. **PeerSim** peut également fonctionner en deux modes différents : par cycle ou par événement. La documentation est fournie sous la forme de Java docs, des tutoriels et un guide de mise en œuvre des protocoles.

PeerSim est sous le permis ouvert de source de GPL et est en partie développé dans le Projet de BISON. PeerSim est écrit dans le langage Java et est principalement maintenu par Mark Jelasity, Gian Paolo Jesi, Alberto Montresor et Spyros Voulgaris de l'université de Bolonia (en Italie) **[7]**.

2. Les modes de simulation avec PeerSim

PeerSim est basé sur deux modes, qui sont tout à fait différents.

2.1. Le mode par cycle (cycle-basé): est basé sur des cycles. À chaque cycle, le simulateur passe par chaque nœud du réseau et exécute chaque protocole associé à ce nœud. Des commandes sont également exécutées périodiquement pour commander la simulation. Elle est basée sur la classe *CDSimulator* du paquet *de peersim.cdsim* **[7]**.

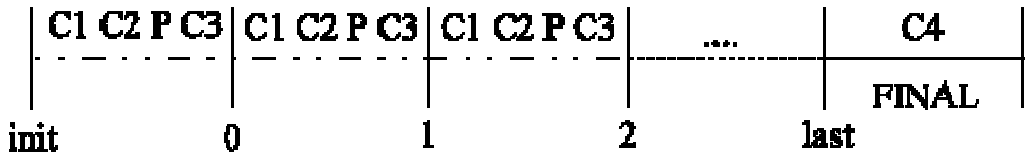

Figure II.4 : mode par cycle

Contrôles : **C** Protocoles : **P**

2.2. Le mode par évènements (événement-basé) : le moteur basé par événements a une manière différente de programmer des événements. Au lieu de programmer l'exécution des différents protocoles avec des cycles, ils sont programmés par des événements. Des événements (ou les messages) sont envoyés aux différents protocoles (par les composants de commande, ou par les protocoles eux-mêmes), et les protocoles peuvent manipuler ces messages et répondre à eux en conséquence. Elle est basée sur la classe *EDSimulator* du paquet de *peersim.edsim*. Étant donné qu'il se fonde sur des messages, le simulateur entraîné par les événements peut émuler une couche transport, de ce fait ajoutant plus de réalisme aux simulations **[7]**.

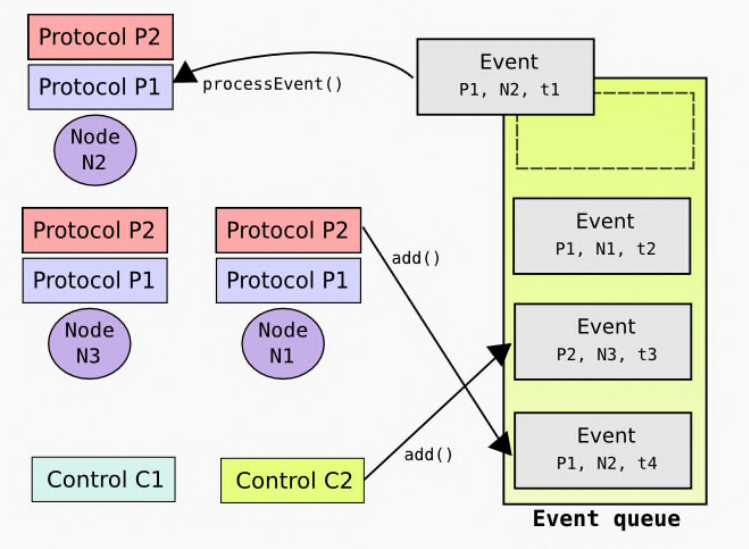

Figure II.5 : mode par évènements

3. Architecture PeerSim

La figure 6 ci-dessous montre l'architecture générale du simulateur **PeerSim** avec exemple de simulation:

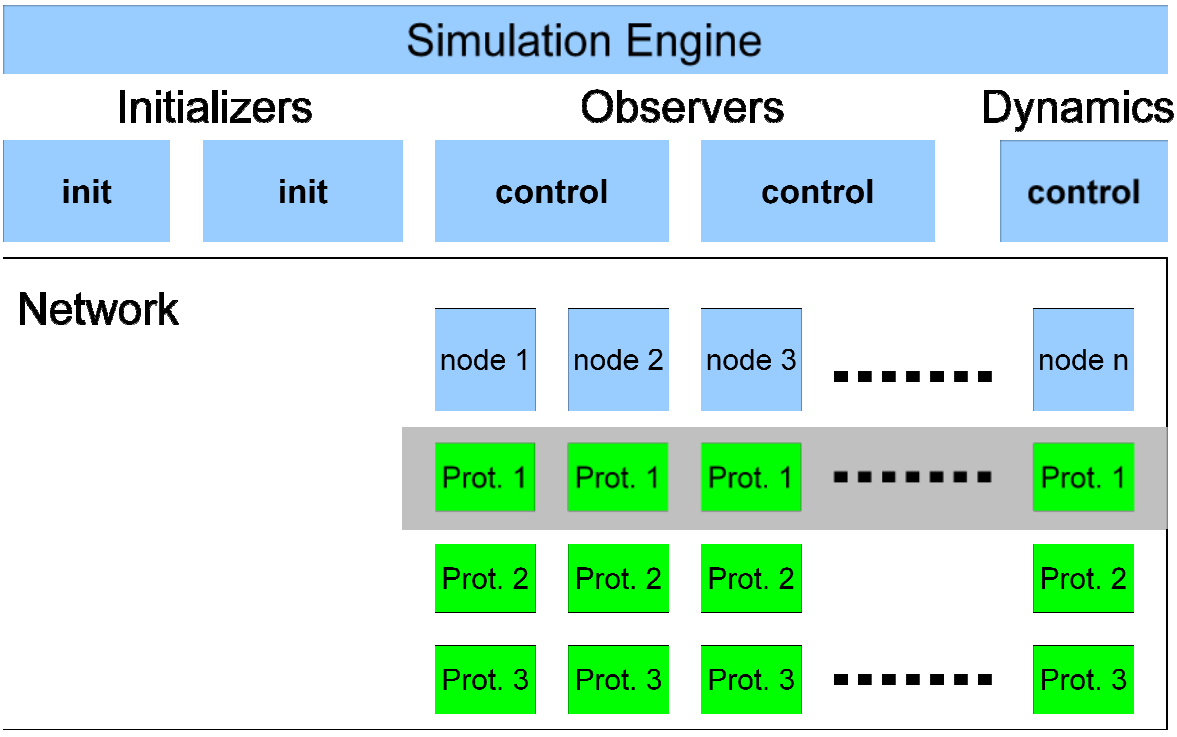

Figure II.6 : Architecture PeerSim avec exemple de simulation

Afin de simuler et observer tout type de réseau P2P, les développeurs de **PeerSim** mettent ainsi à notre disposition trois types de composants : les protocoles, les dynamiques et les observateurs.

3.1. Les protocoles

Dans un réseau P2P que l'on veut simuler avec **PeerSim**, chaque nœud dispose d'une pile de protocoles qui vont être exécutés à chaque cycle de simulation, et ceci dans leur ordre de déclaration dans le fichier de configuration. Bien évidemment, tout est mis à notre disposition pour créer nos propres protocoles.

Chaque protocole est décrit dans la classe java qui porte son nom. Pour les simulations par cycles, celles qui nous intéressent, cette classe doit implémenter l'interface *CDProtocol* (pour *Cycle Driven Protocol*) fournie dans PeerSim. Elle doit donc redéfinir les méthodes *clone()* et *nextCycle(Node, int)*.

Lors du lancement du simulateur, **PeerSim** crée un squelette de nœud avec une instance de tous les protocoles déclarés dans le fichier de configuration, puis clone ce squelette autant de fois que le nombre de nœuds dans le réseau. C'est à cela que sert cette première fonction *clone()*. Il faut à ce propos faire attention à bien réinitialiser les attributs des protocoles dans cette fonction, sinon, lors du clonage, les références des attributs vont être clonées et tous les protocoles auront, dans les faits, les mêmes attributs.

Quant à la fonction *nextCycle(Node, int)*, c'est elle qui définit le comportement du protocole que l'on souhaite implémenter. Elle prend en paramètre le nœud sur lequel se trouve le protocole, et un entier identifiant le numéro du protocole dans la pile de protocole du nœud squelette. C'est cette fonction qui sera appelée à chaque cycle et qui simulera l'exécution de ce protocole sur le nœud courant.

3.2. Les Dynamiques

Leur but est d'introduire un dynamisme dans le réseau, essentiellement sous deux formes : d'une part initialiser le réseau, c'est à dire initialiser les protocoles présents sur les nœuds du réseau, et d'autre part introduire des biais en cours de simulation, par exemple des suppressions de liens entre des nœuds, ou des suppressions de nœuds.

Tout comme pour les protocoles, nous pouvons créer nos propres dynamiques. Il suffit de créer une classe Java implémentant l'interface Dynamiques fournie dans PeerSim, et qui doit donc redéfinir la méthode *modify()*. C'est cette méthode qui sera appelée par le simulateur lors de l'exécution d'une dynamique qui est censé modifier l'état du réseau.

Le fichier de configuration peut contenir deux types de fichiers de dynamiques :

- Le type **initializer** sera exécuté une unique fois avant le début de la simulation.
- Le type **standard** sera exécuté au début de chaque cycle de simulation.

3.3. Les observateurs

Comme leur nom l'indique, ils ont pour but de surveiller le réseau. Ils sont exécutés au début de chaque cycle de simulation et peuvent être utilisés pour de multiples traitements : affichage brut de l'état d'un nœud, collection de données pour un affichage résumé ultérieur, calcul d'indicateurs sur l'état du réseau, statistiques sur ces indicateurs, ...

De la même manière que les deux autres composants, **PeerSim** nous facilite le rajout d'un observer : c'est toujours une classe Java, mais qui implémente cette fois-ci l'interface Observer. Nous devons donc redéfinir la méthode *analyze()* qui sera appelée par le simulateur lors de l'exécution de notre observer **[7]**.

4. Les composants principaux du simulateur PeerSim

PeerSim possède plusieurs packages à savoir :

4.1. Package peersim

L'entrée principale du simulateur PeerSim, le package contient une seule classe qui possède la méthode **'main'** qui charge la configuration et exécute les expériences.

4.2. Package cdsim

Les classes et les interfaces les plus importantes dans ce package sont :

- **Interface CDProtocol :** elle définit les protocoles du cycle, ces protocoles ont une activité périodique dans des intervalles de temps réguliers.
- **Classe CDSimulator :** C'est le moteur de simulation par cycle, il s'agit d'une classe singleton entièrement statique. Pour une simulation par cycle la configuration peut décrire un ensemble de protocoles et leur ordre, un ensemble de contrôles et de leur commande et un ensemble d'initialiseurs et leur classement. Ce moteur ne s'exécute que ceux qui implémentent l'interface CDProtocol.

4.3. Package config

C’est le package qui fait la gestion de la configuration, parmi ses principales classes sont :

- **Classes CheckConfig :** C'est l'outil utilitaire qui vérifie si un fichier de configuration peut être chargé ou non, sans réellement effectuer la simulation.
- **Classe Configuration :** Classe entièrement statique pour stocker des informations de configuration.
- **Classes ConfigProperties :** C’est la classe qui gère les fichiers de configuration.

4.4. Package Core

C’est le package qui contient les classes et les interfaces de base du simulateur, parmi les plus importantes :

- **Interface Control :** Interface générique pour les classes qui sont responsables de l'observation ou la modification de la simulation en cours.
- **Interface Node :** Interface qui représente un nœud avec une adresse de réseau.
- **Interface Protocol :** Interface pour identifier les protocoles.
- **Classe GeneralNode :** Il s'agit de la classe **Node** par défaut qui est utilisé pour composer le réseau.
- **Classe Network :** Cette classe constitue le cadre de base de toutes les simulations.

4.5. Package Dynamics

Il contient des classes de contrôle pour l’initialisation et la modification d’un réseau pendant la simulation.

- **Classe WireKOut :** Cette classe prend un protocole et ajoute des connections aléatoires.
- **Classe DynamicNetwork :** Ce contrôle peut changer la taille des réseaux en ajoutant et en supprimant des nœuds.

5. Fonctionnement générale

Le simulateur **PeerSim** est un ensemble de classes java qui implémentent un simulateur de réseau pair à pair. Deux types de simulations sont possibles, mais celle qui nous intéresse ici est la simulation par cycles : nous allons laisser évoluer le réseau pendant un nombre prédéfini de cycles. Il nous suffira d'analyser l'état du réseau à la fin de la simulation pour en tirer les bonnes conclusions.

Toutes les données utiles au simulateur sont regroupées dans un fichier de configuration où l'on définit les variables du réseau, déclare la structure d'un nœud (les protocoles qui vont s'y exécuter à chaque cycle), les protocoles, les dynamiques et les observateurs que l'on veut utiliser **[8]**.

6. Le fichier de configuration

Chaque simulation est configurée à l'aide d'un fichier de configuration qui définit les composants qui seront utilisés dans la simulation et la manière dont ils interagissent les unes avec les autres. Le fichier de configuration est un simple fichier ASCII. Chaque commentaire est précédé d'un # au début de la ligne. Il n'y a pas d'ordre dans le fichier de configuration pour les instructions, mais la **figure 7** montre comment ordonner le fichier **[7]**.

```
1   ## Global simulation properties
2   random.seed                 1234567890
3   simulation.cycles           1000
4   control.shf                 Shuffle
5   network.size                100000
6
7   ## Protocols
8   # Select a topology initializer
9   protocol.lnk                IdleProtocol
10  # Set the request manager protocol
11  protocol.avg                example.aggregation.AverageFunction
12  protocol.avg.linkable       lnk
13
14  ## Initialisations
15  # for the network topology
16  init.rnd                    WireKOut
17  init.rnd.protocol  lnk
18  init.rnd.k                  20
19  # distribution des valeurs sur les noeuds
20  init.peak                   example.aggregation.PeakDistributionInitializer
21  init.peak.value             10000
22  init.peak.protocol          avg
23
24  ## Controls
25  control.avgo                example.aggregation.AverageObserver
26  control.avgo.protocol       avg
```

Figure II.7 : Exemple de fichier de configuration.

Dans la figure ci-dessus, nous pouvons voir la configuration de la simulation d'un algorithme de moyenne qui sera exécuté en mode par cycle pour un réseau avec 100.000 nœuds (ligne 5) et pour 1000 cycles (ligne 3). On reconnaît les protocoles qui sont dans la l'exemple montrant la pile de protocole. La simulation d'une topologie de réseau de base avec IdleProtocol de protocole (ligne 9), qui ne fait rien sauf de manipuler les connexions entre chaque nœud. Sur chacun des nœuds et à chaque cycle, un algorithme mis en œuvre par AverageFunction (line 11), calcule la moyenne entre la valeur figurant dans ce nœud et une de ses voisines. AverageFunction est en fait une classe de PeerSim qui peut être trouvé dans le répertoire ~ */exemple /agrégation.*

Les composants qui initialisent la simulation sont définis dans la ligne 14. WireKOut à La ligne 16 gère le câblage de nœuds et le degré de la courbe (ligne 18). Ensuite, les valeurs sur les différents nœuds sont initialisées avec la PeakDistributionInitializer composant (ligne 20).

Enfin, un contrôle de la simulation, AverageObserver (ligne 25) qui réunira les statistiques et les imprimer régulièrement **[7]**.

Dans l'exemple ci-dessous montre le résultat de la **figure 7** :

```
1   Simulator: loading configuration
2   ConfigProperties: File example/config-test.txt loaded.
3   Simulator: starting experiment 0 invoking peersim.cdsim.CDSimulator
4   Random seed: 1234567890
5
6   CDSimulator: resetting
7   Network: no node defined, using GeneralNode
8   CDSimulator: running initializers
9   - Running initializer init.peak: class example.aggregation.PeakDistributionInitializer
10  - Running initializer init.rnd: class peersim.dynamics.WireKOut
11  CDSimulator: loaded controls [control.avgo, control.shf]
12  CDSimulator: starting simulation
13  control.avgo: 0 0.0 10000.0 100000 0.1 1000.0 99999 1
14  CDSimulator: cycle 0 done
15  control.avgo: 1 0.0 2500.0 100000 0.1 187.49187491874918 99994 2
16  CDSimulator: cycle 1 done
17  control.avgo: 2 0.0 1250.0 100000 0.1 54.06768911439114 99972 1
18  CDSimulator: cycle 2 done
19  control.avgo: 3 0.0 625.0 100000 0.1 19.366948430226486 99848 2
20  CDSimulator: cycle 3 done
21  control.avgo: 4 0.0 312.5 100000 0.1 6.383266107582426 99206 2
22  CDSimulator: cycle 4 done
23  control.avgo: 5 0.0 156.25 100000 0.1 2.0482095720890525 95974 1
24  CDSimulator: cycle 5 done
25  control.avgo: 6 0.0 78.277587890625 100000 0.1 0.75166241219096 81391 2
26  CDSimulator: cycle 6 done
27  control.avgo: 7 0.0 39.1387939453125 100000 0.1 0.239095922144 39078 2
28  CDSimulator: cycle 7 done
29  control.avgo: 8 0.0 19.56939697265625 100000 0.1 0.070958498117 3906 1
30  CDSimulator: cycle 8 done
31  ...
```

Figure II.8 : Premières lignes du résultat de la simulation.

D'abord, nous avons des informations de base comme le nom du fichier de configuration (ligne 2), la semence de génération de nombres aléatoires (ligne 4), le simulateur a utilisé (cycle-based CDSimulator, ligne 3) et les initialiseurs (ligne 9 et 10). Ensuite, nous avons les résultats de la simulation eux-mêmes. Seuls les 8 premiers cycles (sur 10000) sont présentés. Aux lignes 13, 15, 17, 19 et ainsi de suite, nous pouvons voir la sortie de l'AverageObserver (AVGo), le contrôle que nous avons vu précédemment dans le fichier de configuration **[7]**.

Conclusion

Dans ce chapitre nous avons parlé sur le simulateur de réseau pair à pair **PeerSim**. Il est caractérisé par sa dynamicité et extensibilité. Grâce à son approche modulaire nous pouvons ajouter et réutiliser les modules existants, c'est pour ces raisons qu'il est l'un des plus connu, parce qu'il facilite grandement le codage de nouvelle application.

Dans le chapitre suivant nous allons voir quelques concepts sur la gestion de données dans les réseaux P2P.

Chapitre *III*

Gestion de données

Dans les réseaux P2P

« Il m'a toujours paru
que l'on pouvait recueillir
plus d'instructions de
ce qui se répète que de
ce qui ne recommence jamais. »

A.Gide

Introduction

Ces dernières années, les systèmes pair-à-pair (P2P) sont devenus très populaires. Cette popularité vient des bonnes caractéristiques offertes par ces systèmes comme : passage a l'échelle, l'autonomie des nœuds et le contrôle décentralisé. Les systèmes P2P sont bien adaptés aux environnements distribués à grande échelle dans lesquels les nœuds (appelés indifféremment nœuds ou pairs) peuvent partager leurs ressources **[9]**. Le type de ces sources de données sont variées (données textuelles, relationnelles, multimédia, XML, vues matérialisées) et leurs systèmes de stockage très différents (système de fichiers, SGBD, applications) **[10]**.

1. Modèles de données des systèmes P2P

Nous allons parler de quelques types de données qui circulent dans les réseaux P2P:

1.1. XML

XML offre un mécanisme idéal pour transférer des messages courts et structurés entre les applications par les pairs. Il peut être facilement personnalisé pour les systèmes P2P spécifiques et facilement transmis sur les protocoles de l'Internet d'aujourd'hui. Les données XML peuvent être cryptées en utilisant les technologies existantes, ce qui en fait un candidat idéal pour des messages sécurisés **[11]**.

1.2. Données relationnelles et Objet

Récemment, les systèmes P2P de partage de données sont apparus, d'un côté, comme une nouvelle génération des systèmes P2P et, de l'autre côté, comme un nouveau pas dans le long chemin de recherche en bases de données. Etant donné que le modèle relationnel de bases de données est largement utilisé pour limitation d'étude de ce modèle. Un système P2P de partage de données qui est informellement définit comme un système réparti à grande-échelle dans lequel les nœuds sont autonomes et peuvent se connecter/se déconnecter au/du système à n'importe quel moment d'une façon complètement décentralisée. Chaque nœud a son propre système de bases de données (SBD) qui est constitué d'un SGBD et d'une (ou plusieurs) base(s) de données qu'il gère. Dans un tel environnement, les nœuds jouent des rôles symétriques. Chaque nœud joue le rôle :

- ✓ un client, lorsqu'il consomme de ressources disponibles sur d'autres nœuds dans le système.

- ✓ un serveur, lorsqu'il fournit des services aux autres nœuds comme par exemple le stockage des méta-données concernant ses voisins ou le stockage des relations temporaires lors des exécutions de requêtes.
- ✓ un routeur, lorsqu'il propage des requêtes et des messages vers d'autres nœuds dans le système.
- ✓ un hôte des sources de données, lorsqu'il partage ses données avec d'autres nœuds dans le système.

Chaque nœud gère d'une manière autonome sa propre base de données. Les bases de données ont un intérêt commun. Elles stockent des données similaires avec une interdépendance sémantique. Néanmoins, à cause de l'autonomie de nœuds, les données pourraient être représentées par des schémas locaux hétérogènes. Il n'y a pas de catalogue global ni centralisé ni dupliqué.

Les nœuds désirent partager leurs données d'une façon compréhensible et efficace. Le mot ‘‘compréhensible’’ signifie que lorsqu'un nœud soumet une requête dans le système, les autres nœuds doivent comprendre la sémantique de cette requête afin d'envoyer les données souhaitées et d'éviter l'envoi des réponses invalides. Ces réponses invalides nécessitent une consommation de ressources (e.g. bande passante, capacité de stockage et puissance de calcul) pour aucun bénéfice. Par le mot ‘‘efficace’’, qu'il faut désigner réduire le temps de réponse d'une requête donnée tout en minimisant, le lus possible, la consommation des ressources disponibles lors de l'évaluation de la requête.

L'environnement considéré forme un monde ouvert. C'est-à-dire, un nœud imprévu doit être capable d'entrer dans le système sans aucune administration centralisée. L'utilisateur de chaque nœud doit être capable de modifier facilement la partie partagée de ses données. Il doit être capable, aussi, de choisir pour une requête soumise si cette requête doit être exécutée localement en utilisant le SBD local ou globalement (en permettant aux autres nœuds de participer à l'évaluation de sa requête) **[9]**.

1.3. Multimédia

Dans le milieu universitaire et l'industrie, peer-to-peer (P2P) ont attiré une grande attention. Les applications de partage de fichiers, tels que Napster, Gnutella, Kazaa, BitTorrent, Skype et PPLive Peer-to-peer, ont assisté à un énorme succès parmi les utilisateurs finaux. Et les utilisations de réseau peer-to-peer pour le streaming multimédia, conférences, jeux, sauvegarde de fichiers, la recherche d'information est à la hausse.

Statistiques récentes montrent que le trafic P2P pour autant que 70% du trafic Internet. Contrairement à un système client-serveur, les pairs apportent avec eux servant capacité.

Par conséquent, comme la demande d'un système Peer-to-peer augmente, la capacité du réseau se développe, aussi. Cela permet une application multimédia peer-to-peer pour pas cher à construire et superbe en termes d'évolutivité. Les technologies peuvent être appliquées au partage P2P de fichiers, conférence de P2P, P2P streaming media, P2P VoIP et les applications de stockage de P2P **[11]**.

2. Réplication de données dans les systèmes P2P

Réplication peer-to-peer fournit un scale-out et solution de haute disponibilité en maintenant des copies de données entre plusieurs instances de serveur, aussi appelé nœuds. Construit sur la base de la réplication transactionnelle, la réplication peer-to-peer propage les modifications cohérence transactionnelle en temps quasi-réel. Cela permet des applications qui nécessitent une montée en puissance d'opérations de lecture à distribuer les lectures de clients sur plusieurs nœuds. Parce que les données sont conservées sur les nœuds en temps quasi réel, la réplication peer-to-peer fournit une redondance de données, ce qui augmente la disponibilité des données **[14]**. Par exemple, dans une base de données relationnelle répliqué, si les tables sont entièrement reproduits ensuite tableaux correspondent à des objets, mais si il est possible de reproduire tuples individuels, alors tuples correspondent à des objets. D'autres exemples d'objets comprennent des documents XML, des fichiers typés, des fichiers multimédias, etc.

La réplication de données est très importante dans le contexte de systèmes distribués pour plusieurs raisons. Tout d'abord, la réplication améliore la disponibilité du système en supprimant les points uniques de défaillance (objets sont accessibles à partir de plusieurs sites). Deuxièmement, elle améliore la performance du système en réduisant le temps système de communication (objets peuvent être situés plus près de leurs points d'accès) et d'augmenter le débit du système (plusieurs sites servent le même objet en même temps). Enfin, la réplication permet d'améliorer la modularité du système, car il supporte la croissance du système avec un temps de réponse acceptable **[12]**.

3. Les vues matérialisées

Une vue est une table virtuelle représentant le résultat d'une requête sur la base. Comme son nom l'indique et à la différence d'une vue standard, dans une **vue matérialisée** les données de la vue sont stockées dans la base. On l'utilise essentiellement à des fins d'optimisation et de performance dans le cas où la requête associée est particulièrement complexe ou lourde, ou pour faire des réplications de table. La fraîcheur des données de la vue matérialisée dépend des options choisies lors de sa création. Le décalage entre les données de la table maître et la vue matérialisée peut être nul (rafraîchissement synchrone) ou d'une durée planifiée : heure, jour, etc. Suivant le contexte il existe différents types de vue matérialisée possibles : sur clé primaire, rowid (identifiant unique des tuples), et plus ou moins complexes : avec fonctions d'agrégation, sous-requêtes, jointures, etc.

Les vues matérialisées sont employées depuis des décennies dans les bases de données afin de raccourcir les temps de traitement des requêtes. Elles peuvent être considérées les résultats de requêtes pré-calculées, que l'on réutilise afin d'éviter de recalculer (complètement ou partiellement) une nouvelle requête. Les vues matérialisées ont fait l'objet de nombreuses recherches, en particulier dans le contexte des entrepôts des données relationnelles **[12]**.

Ces dernières années, les vues ont trouvé de nouvelles applications. Elles permettent de définir des tables virtuelles correspondant aux besoins des programmes d'application en termes de données **[13].**

Conclusion

Au cours de ce chapitre, nous nous somme rapproché de l'environnement technique de notre projet, et cela nous a fait comprendre quelques concepts de notre projet pour mieux répondre à ses besoins.

Dans le chapitre suivant nous commençant l'analyse des besoins et la conception de notre projet.

Chapitre *IV*

Analyse et conception

« Comprendre, c'est traduire. »

George Steiner

Introduction

L'analyse et la conception sont deux étapes importantes. La première consiste à étudier précisément la spécification fonctionnelle de manière a obtenir une idée sur ce que va réaliser le système, la deuxième définit l'architecture globale du système aux niveaux logique et physique. Dans ce chapitre nous allons présenter la problématique puis notre solution, ainsi que l'analyse dans laquelle nous allons présenter la solution détaillée de notre système, dans la conception nous allons définir l'architecture globale du système, en utilisant le langage UML et le processus 2TUP.

1. Problématique

Les réseaux Peer-To-Peer représentent en fonction des régions de notre globe de 50 à 90% du trafic Internet. Un tel engouement pour les applications P2P, peut s'expliquer par les avantages multiples que présentent ces systèmes, qui permettent principalement une mutualisation des ressources informatiques (Informations, espace de stockage ou puissance CPU…).

Le grand défi de nos jours dans le domaine des systèmes distribués est d'exploiter le potentiel des réseaux P2P non plus seulement pour diffuser du contenu mais aussi pour créer et éditer ce contenu. En effet, les outils de travail collaboratif sont de plus en plus répandus, et certaines de ces applications permettent à des utilisateurs situés à des endroits géographiquement éloignés de travailler, consulter ou modifier un document commun. La fonction primordiale, ou la propriété clé de ces systèmes, est d'assurer la gestion des documents et la cohérence des données à la fin de chaque processus.

Afin de tester et simuler ces systèmes, il faut choisir un simulateur rapide, extensible, portable, maintenu, il existe plusieurs simulateurs qui répondent à ces caractéristiques, **PeerSim** est l'un des simulateurs le plus utilisée parmi les chercheurs et les développeurs, Il a l'avantage d'être déjà spécialisé pour l'étude des systèmes P2P et qui dispose d'une architecture ouverte, et une approche modulaire qui permet de l'adapter et de le spécialiser.

Notre problème est donc de pouvoir concevoir et implémenter un module de gestion de données pour PeerSim, qui permet de gérer les différents types de données qui circulent dans les réseaux P2P, et pouvoir créer et initialiser des requêtes a travers le réseau P2P, en mettant en place des mécanismes de gestion de données. Notre module permettra de créer et mettre à jour la BDD, créer et répondre aux requêtes.

2. Objectif du projet

Les objectifs du futur système sont :

- Permet l'ajout, la modification et la suppression d'une donnée (xml, tables, vues).
- Permet la gestion des requêtes telle qu'un utilisateur peut créer et initialiser des requêtes.
- Permettre la création et la mise à jour d'une base de données.
- Permet de récolter des informations sur l'état du réseau au cours de la simulation.

3. Analyse

3.1. Capture des besoins fonctionnels

La capture des besoins fonctionnels, produit un modèle des besoins focalisé sur le métier des utilisateurs. Elle qualifie au plus tôt le risque de produire un système inadapté aux utilisateurs **[15]**.

Les différentes étapes de la partie fonctionnelle sont les suivants :

3.1.1. Identification des acteurs

Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié. Un acteur peut consulter et/ou modifier directement l'état du système, en émettant et/ou en recevant des messages susceptibles d'être porteurs de données **[16]**.

Dans notre étude, le seul acteur qui interagie avec le système est l'utilisateur qui simule avec PeerSim.

L'utilisateur peut **:**

- ajouter, modifier et supprimer une requête.
- créer et mettre à jour une base de données.
- ajouter, modifier et supprimer un document XML.
- ajouter, modifier et supprimer une vue.

3.1.2. Identification des cas d'utilisation

Un cas d'utilisation permet de décrire l'interaction entre les acteurs (utilisateurs du cas) et le système. La description de l'interaction est réalisée suivant le point de vue de

l'utilisateur. La représentation d'un cas d'utilisation met en jeu trois concepts : l'acteur, le cas d'utilisation et l'interaction entre l'acteur et le cas d'utilisation **[17]**.

En ce qui concerne notre application, nous allons décrire les différents cas d'utilisation du système à réaliser :

Acteurs	**Cas d'utilisation**
Utilisateur	Gestion des requêtes Gestion des documents XML Gestion des tables Gestion des vues

Tableau IV.2 : Identification des cas d'utilisation.

3.1.3. Description textuelle des cas d'utilisation

Le **Tableau IV.3** montre la description textuelle de cas d'utilisation « **Gestion des requêtes** ».

Titre	Gestion des requêtes
But	Permet à l'utilisateur de gérer les requêtes (Ajouter, modifier et supprimer une requête)
Acteurs	Utilisateur
Scénario nominal	- L'utilisateur prépare le fichier de configuration pour l'ajout, la modification ou la suppression d'une requête. - L'utilisateur lance la simulation sur l'IDE java. - Le simulateur vérifie le fichier de configuration ✓ 1ère vérification : le simulateur vérifie la syntaxe de fichier de configuration. ✓ 2ème vérification : le module de gestion de données vérifie l'existence de la requête. - Le simulateur affiche le résultat de la simulation.
Scénario alternatif	- S'il y a une erreur dans le fichier de configuration, le simulateur affiche un message d'erreur syntaxique. - Si la requête ne peut pas être ajoutée, modifiée ou supprimée le module de gestion de données affiche un message d'erreur.

Tableau IV.3 : Description textuelle du cas d'utilisation « **Gestion des requêtes** ».

Le **Tableau IV.4** montre la description textuelle de cas d'utilisation « **Gestion des documents XML** ».

Titre	Gestion des documents XML
But	Permet à l'utilisateur de gérer les documents XML (Ajouter, modifier et supprimer un document XML)
Acteurs	Utilisateur
Scénario nominal	- L'utilisateur prépare le fichier de configuration pour l'ajout, la modification ou la suppression d'un document XML. - L'utilisateur lance la simulation sur l'IDE java - Le simulateur vérifie le fichier de configuration ✓ 1ère vérification : le simulateur vérifie la syntaxe de fichier de configuration. ✓ 2ème vérification : le module de gestion de données vérifie l'existence du document XML. - Le simulateur affiche le résultat de la simulation
Scénario alternatif	- S'il y a une erreur dans le fichier de configuration, le simulateur affiche un message d'erreur syntaxique. - Si l'ajout, modification ou la suppression d'un document XML ne se fait pas le module de gestion de données affiche un message d'erreur.

Tableau IV.4 : Description textuelle du cas d'utilisation « **Gestion des documents XML**».

Le **Tableau IV.5** montre la description textuelle de cas d'utilisation « **Gestion des tables** ».

Titre	Gestion des tables
But	Permet à l'utilisateur de gérer les tables (relationnelle, objet)
Acteurs	Utilisateur
Scénario nominal	- L'utilisateur prépare le fichier de configuration pour gérer les tables. - L'utilisateur lance la simulation sur l'IDE java - Le simulateur vérifie le fichier de configuration ✓ 1ère vérification : le simulateur vérifie la syntaxe de fichier de configuration. ✓ 2ème vérification : le module de gestion de données vérifie l'existence de la table. - Le simulateur affiche le résultat de la simulation

Scénario alternatif	- S'il y a une erreur dans le fichier de configuration, le simulateur affiche un message d'erreur syntaxique. - Si l'ajout, modification ou la suppression d'une table ne se fait pas le module de gestion de données affiche un message d'erreur.

Tableau IV.5 : Description textuelle du cas d'utilisation « **Gestion des tables**».

Le **Tableau IV.6** montre la description textuelle de cas d'utilisation « **Gestion des vues** ».

Titre	Gestion des vues
But	Permet à l'utilisateur de gérer les vues
Acteurs	Utilisateur
Scénario nominal	- L'utilisateur prépare le fichier de configuration pour gérer les vues. - L'utilisateur lance la simulation sur l'IDE java - Le simulateur vérifie le fichier de configuration ✓ 1ère vérification : le simulateur vérifie la syntaxe de fichier de configuration. ✓ 2ème vérification : le module de gestion de données vérifie l'existence de la vue. - Le simulateur affiche le résultat de la simulation
Scénario alternatif	- S'il y a une erreur dans le fichier de configuration, le simulateur affiche un message d'erreur syntaxique. - Si l'ajout, modification ou la suppression d'une vues ne se fait pas le module de gestion de données affiche un message d'erreur.

Tableau IV.6 : Description textuelle du cas d'utilisation « **Gestion des vues**».

3.1.4. Diagramme de cas d'utilisation associé à l'utilisateur

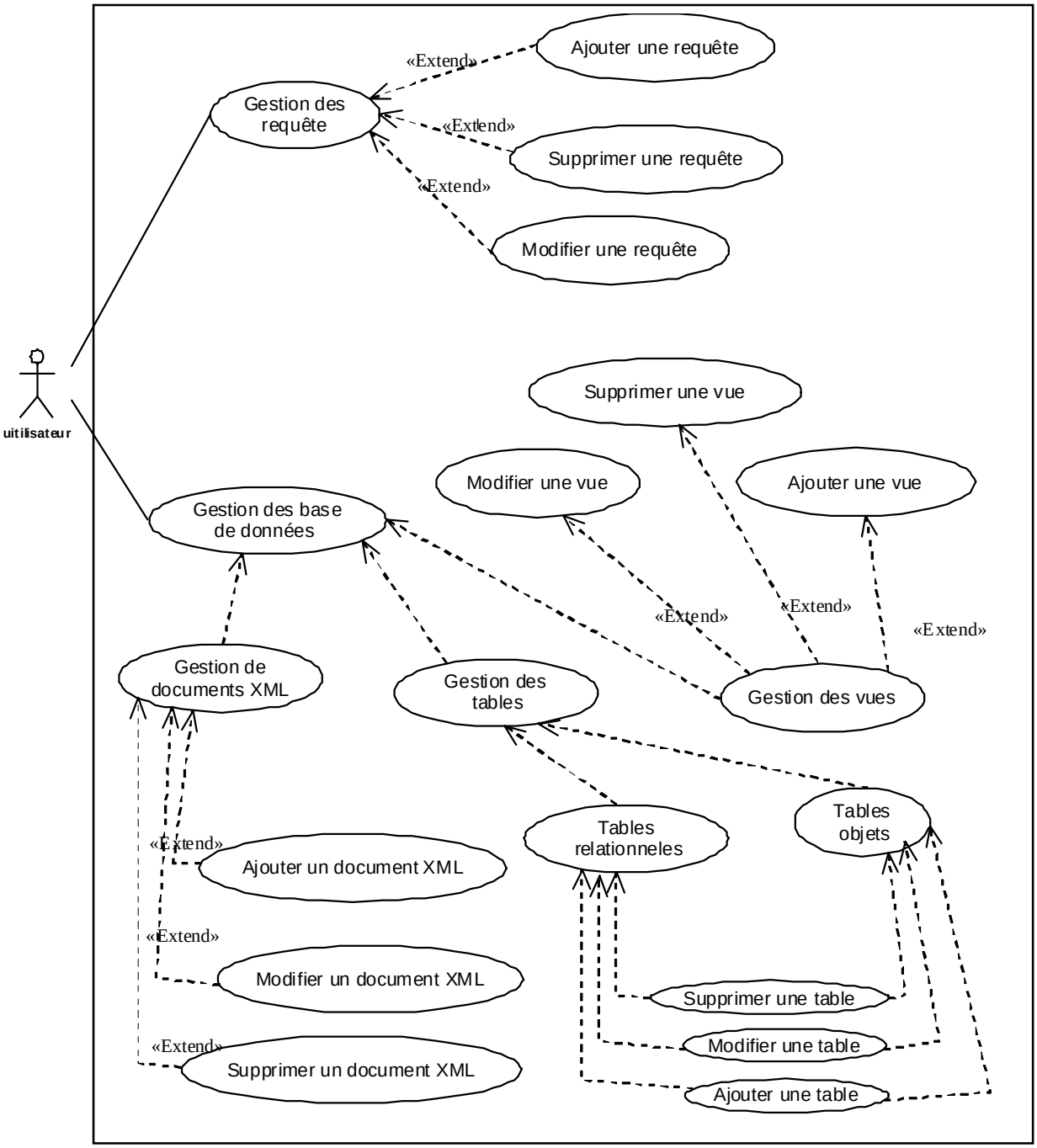

Figure IV.9 : Diagramme de cas d'utilisation associé à l'utilisateur.

3.1.5. Diagramme de séquence

L'objectif du **diagramme de séquence** est de représenter les interactions entre objets en indiquant la chronologie des échanges. Cette représentation peut se réaliser par cas d'utilisation en considérant les différents scénarios associés **[17]**.

3.1.5.1. Message synchrone et asynchrone

Dans un diagramme de séquence, deux types de messages peuvent être distingués : **[17]**

- **Message synchrone :** Dans ce cas l'émetteur reste en attente de la réponse à son message avant de poursuivre ses actions. La flèche avec extrémité pleine symbolise ce type de message. Le message retour peut ne pas être représenté car il est inclus dans la fin d'exécution de l'opération de l'objet destinataire du message.
- **Message asynchrone :** Dans ce cas, l'émetteur n'attend pas la réponse à son message, il poursuit l'exécution de ses opérations. C'est une flèche avec une extrémité non pleine qui symbolise ce type de message.

3.1.5.2. Fragment d'interaction

Dans un diagramme de séquence, il est possible de distinguer des sous-ensembles d'interactions qui constituent des fragments. Un fragment d'interaction se représente globalement comme un diagramme de séquence dans un rectangle avec indication dans le coin à gauche du nom du fragment **[17]**.

Il existe 13 opérateurs définis dans la notation UML 2. Nous citons, dans ce qui suit, les opérateurs utilisés pour l'analyse de notre projet :

- **Opérateur alt (alternative) :** correspond à une instruction de test avec une ou plusieurs alternatives possibles. Il est aussi permis d'utiliser les clauses de type sinon.
- **Opérateur opt :** correspond à une instruction de test sans alternative.
- **Opérateur loop :** correspond à une instruction de boucle qui permet d'exécuter une séquence d'interaction tant qu'une condition est satisfaite.
- **Opérateur ref** : permet d'appeler une séquence d'interaction décrite par ailleurs constituant ainsi une sorte de sous-diagramme de séquence.

3.1.5.3. Les diagrammes de séquences des cas d'utilisation

Dans ce qui suit le terme **«** Vérification **»** dans les diagrammes de séquences signifie qu'il y a deux vérification :

- ✓ La première vérification c'est la syntaxe de fichier de configuration.
- ✓ La deuxième vérification se fait au niveau de la base de données.

3.1.5.3.1. Diagramme de séquence « Ajouter une requête »

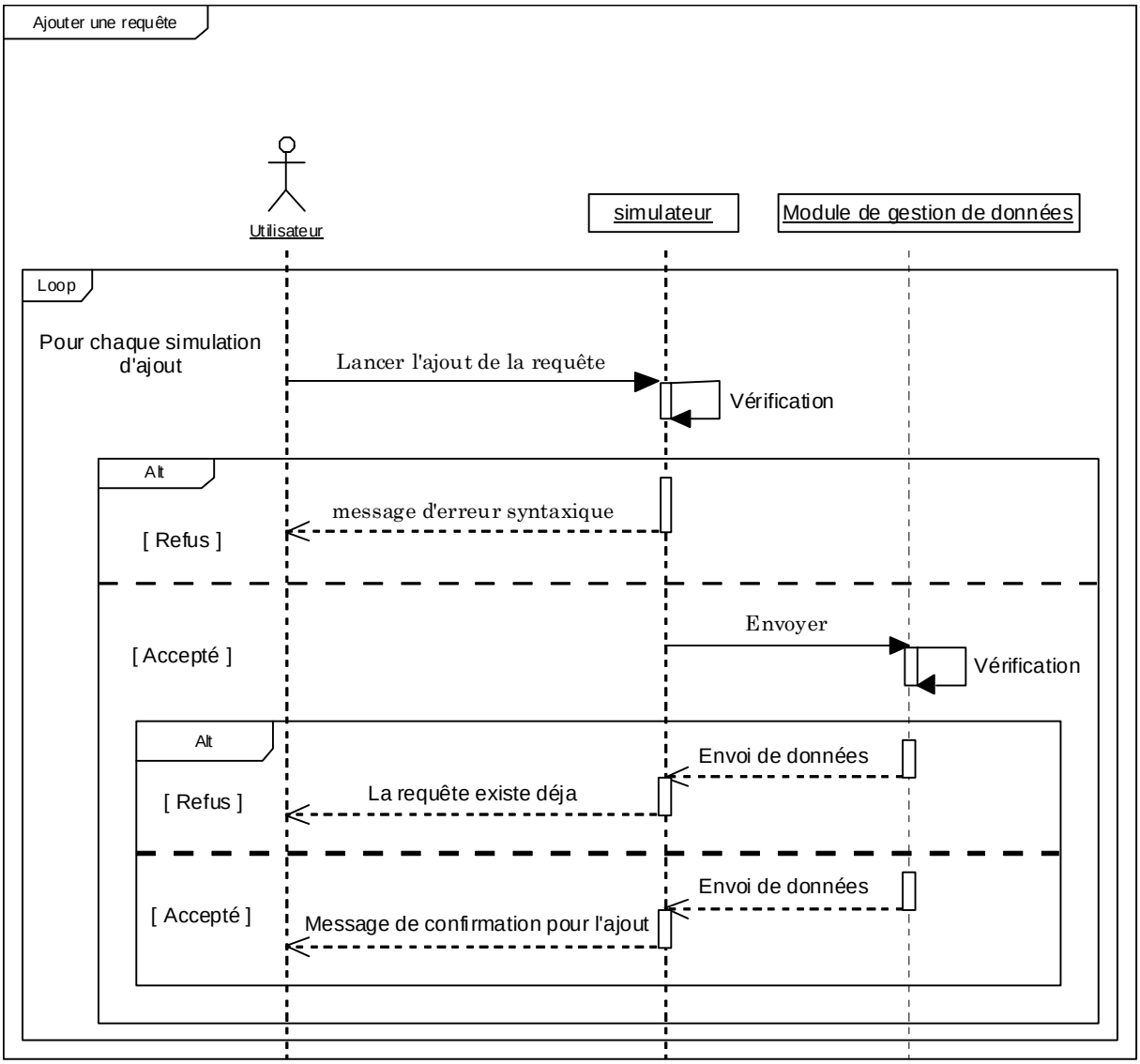

Figure IV.10 : Diagramme de séquence de cas d'utilisation **« Ajouter une requête ».**

3.1.5.3.2. Diagramme de séquence « Modifier une requête »

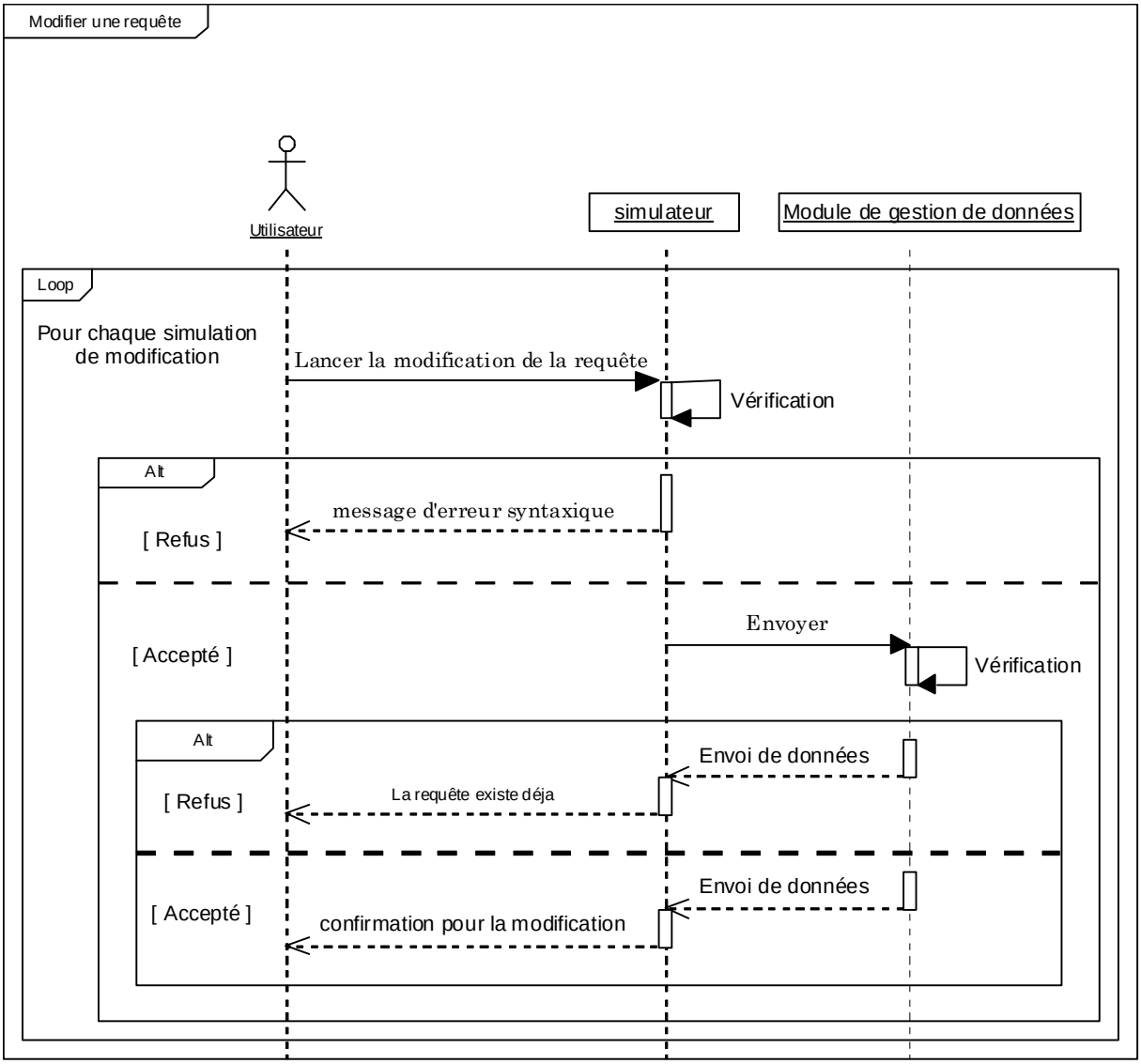

Figure IV.11 : Diagramme de séquence de cas d'utilisation **« Modifier une requête ».**

3.1.5.3.3. Diagramme de séquence « Supprimer une requête »

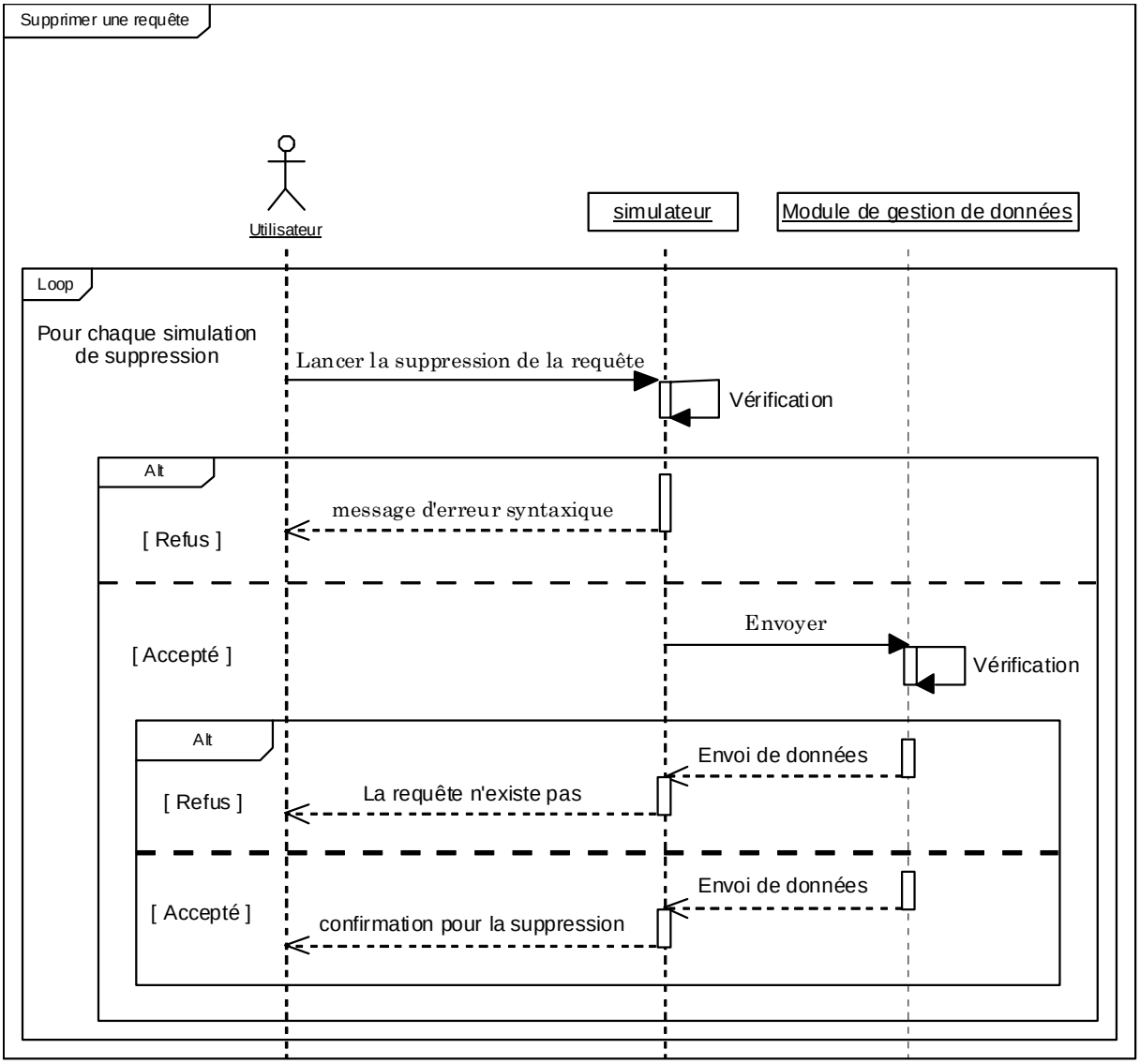

Figure IV.12 : Diagramme de séquence de cas d'utilisation **« Supprimer une requête ».**

3.1.5.3.4. Diagramme de séquence « Gestion des documents XML »

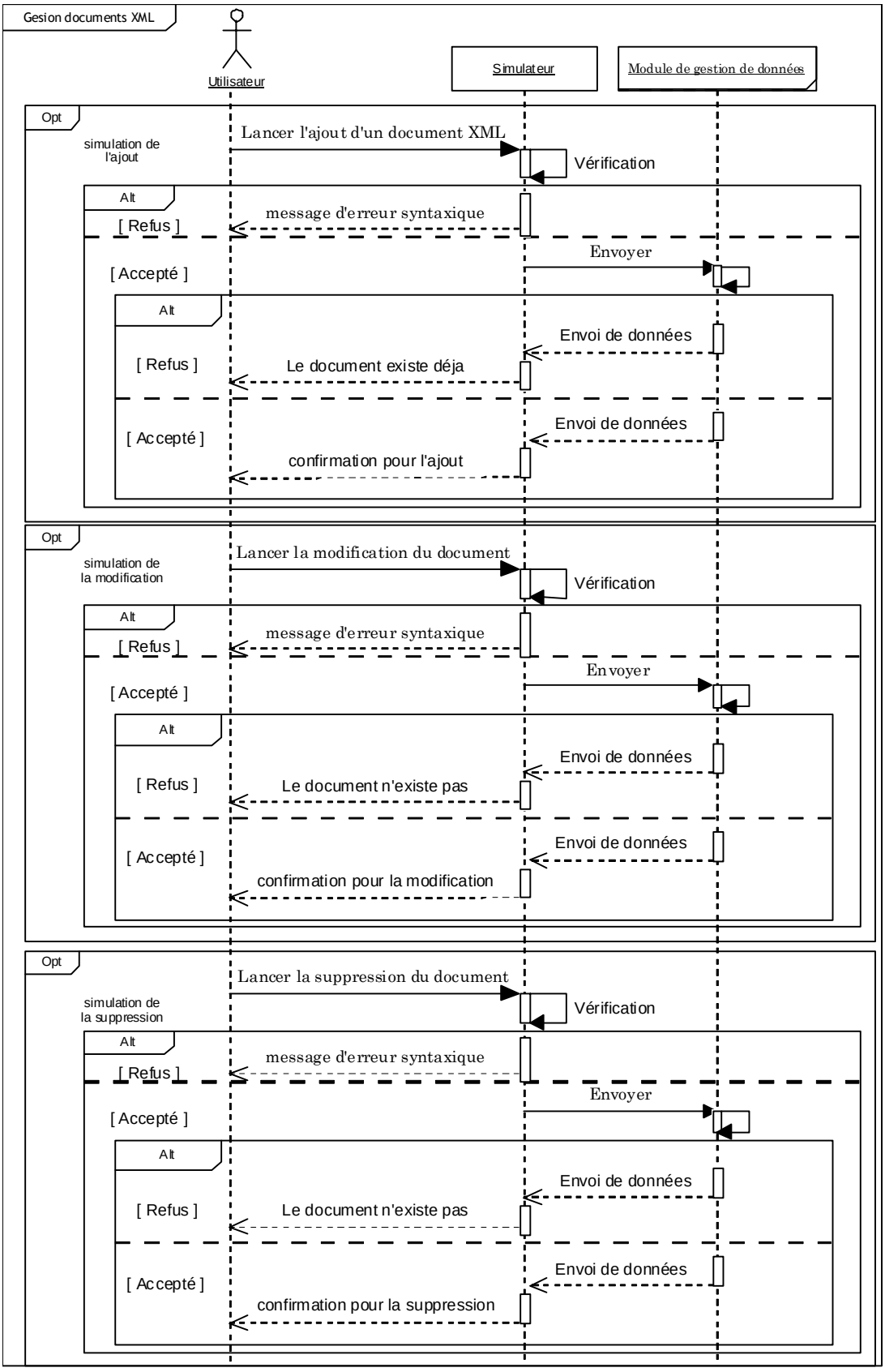

Figure IV.13 : Diagramme de séquence de cas d'utilisation **« Gestion des documents XML ».**

3.1.5.3.5. Diagramme de séquence « Gestion des tables»

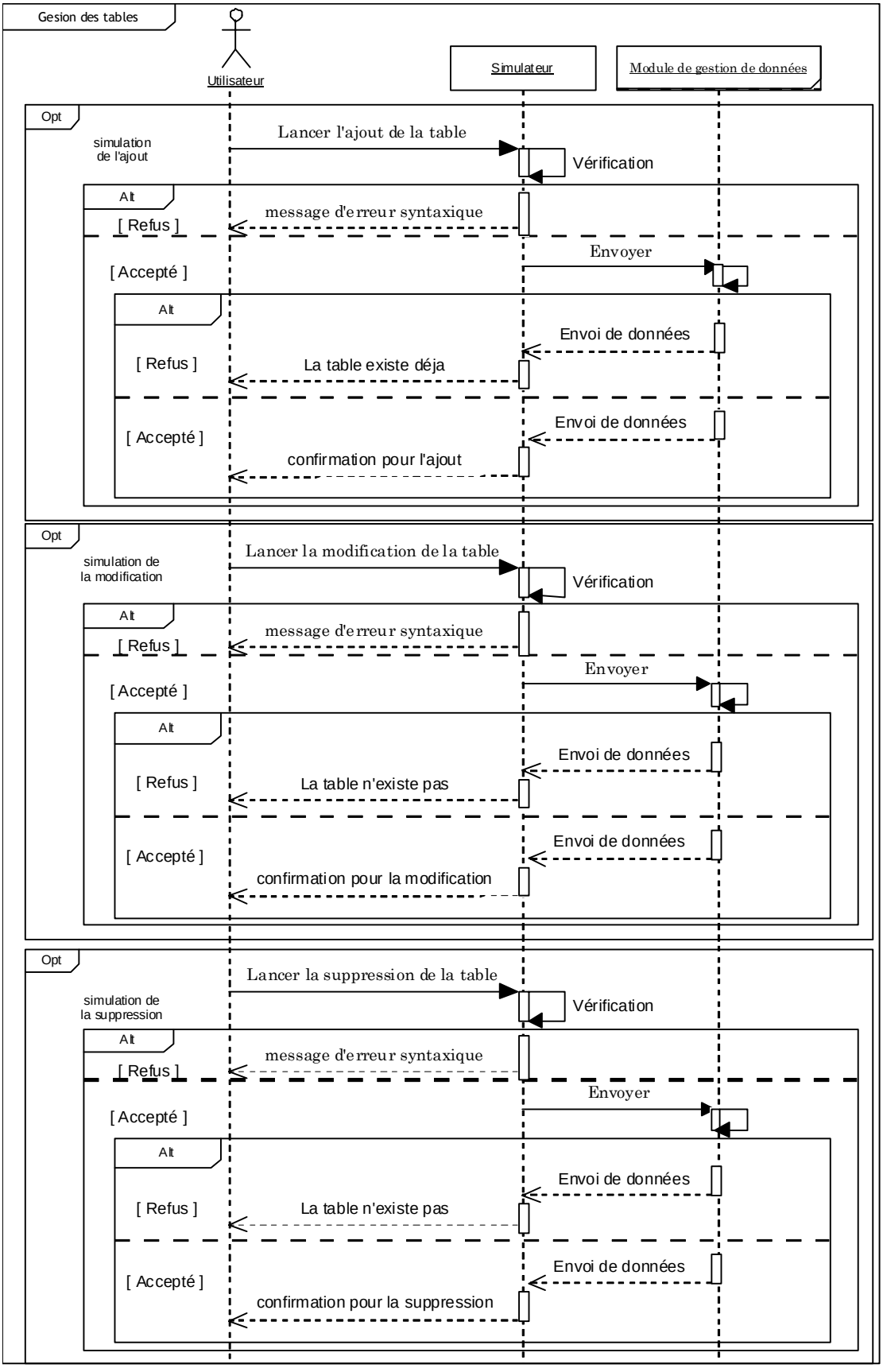

Figure IV.14 : Diagramme de séquence de cas d'utilisation **« Gestion des table ».**

4. Conception

La phase de conception suit immédiatement la phase d'analyse, elle concerne l'interprétation des besoins et la méthode adaptée pour les satisfaire dans l'application, en d'autres termes cette étape consiste à enlever toute abstraction apparue dans l'étape précédente et à donner une vision générale et simple sur le système avant l'implémentation.

4.1. Diagramme de classe

Le diagramme de classe constitue l'un des pivots essentiels de la modélisation avec UML. En effet, ce diagramme permet de donner la représentation statique du système à développer. Cette représentation est centrée sur les concepts de classe et d'association. Chaque classe se décrit par les données et les traitements dont elle est responsable pour elle-même et vis-à-vis des autres classes. Les traitements sont matérialisés par des opérations. Le détail des traitements n'est pas représenté directement dans le diagramme de classe; seul l'algorithme général et le pseudo-code correspondant peuvent être associés à la modélisation **[18]**.

La description du diagramme de classe est fondée sur :

- le concept d'objet.
- le concept de classe comprenant les attributs et les opérations.
- les différents types d'association entre classes.

Un diagramme de classe est composé des éléments suivants : **[19]**

- **Classe :** Une classe représente la description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques. On peut parler également de type.
- **Objet :** Un objet est une entité aux frontières bien définies, possédant une identité et encapsulant un état et un comportement. Un objet est une instance (ou occurrence) d'une classe.
- **Attribut :** Un attribut représente un type d'information contenu dans une classe.
- **Opération :** Une opération représente un élément de comportement (un service) contenu dans une classe.
- **Association** : Une association représente une relation sémantique durable entre deux classes.
- **Agrégation** : Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance.

- **Composition** : Une composition est une agrégation plus forte impliquant que :
  - ✓ un élément ne peut appartenir qu'à un seul agrégat composite (agrégation non partagée).
  - ✓ la destruction de l'agrégat composite entraîne la destruction de tous ses éléments (le composite est responsable du cycle de vie des parties).
- **Généralisation, Super-classe, Sous-classe :** Une super-classe est une classe plus générale reliée à une ou plusieurs autres classes plus spécialisées (sous-classes) par une relation de généralisation. Les sous-classes « héritent » des propriétés de leur super-classe et peuvent comporter des propriétés spécifiques supplémentaires.

4.2. Représentation graphique du diagramme de classes

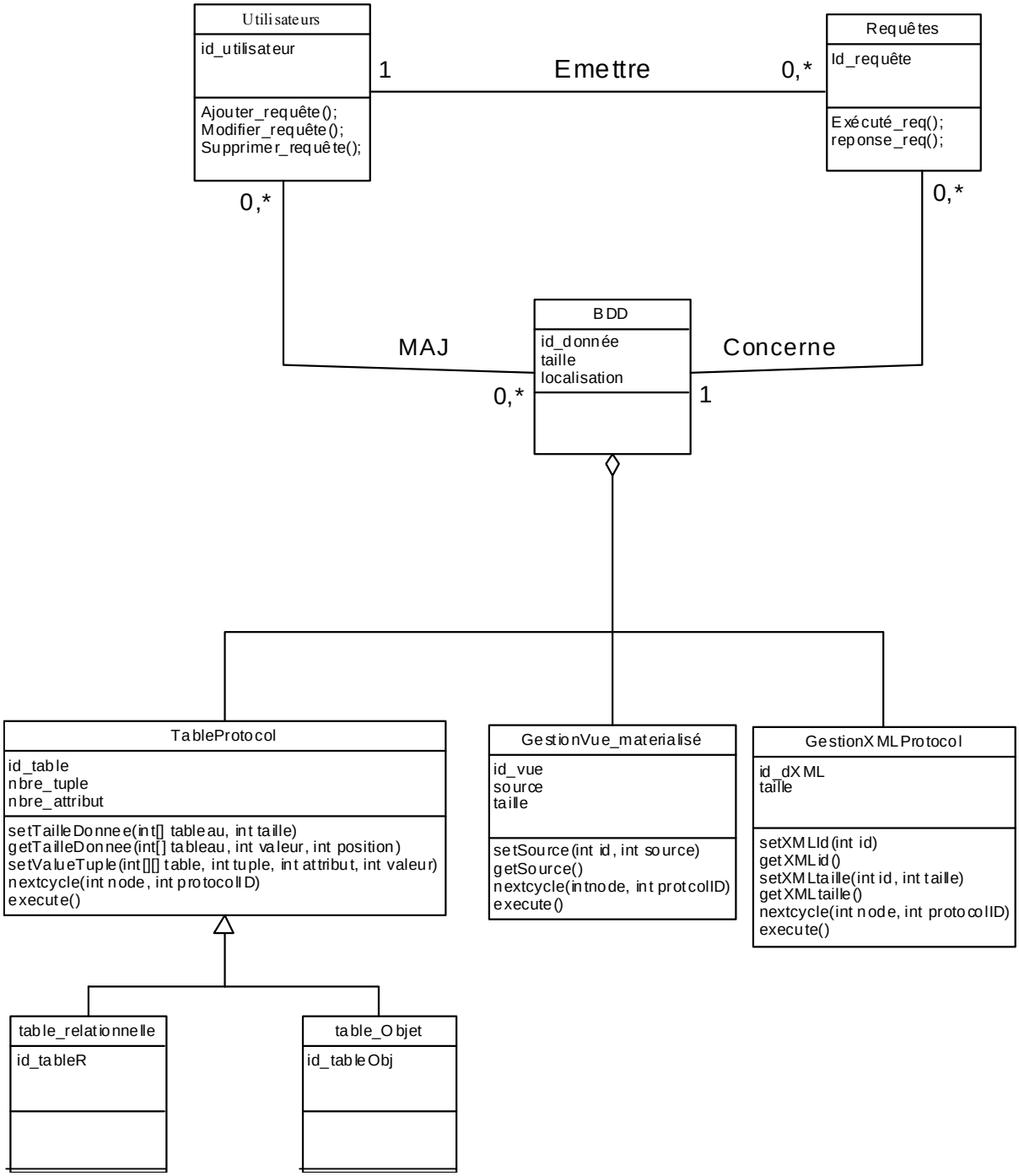

Figure IV.15 : Diagramme de classes.

Conclusion

L'analyse fonctionnelle nous a permis d'avoir une vision prématurée sur le métier à étudier avec la définition des différents acteurs subsistants et les diagrammes de cas d'utilisation pour illustrer les interactions avec le système, puis le séquencèrent des cas d'utilisation à travers des diagrammes de séquences. Enfin, nous avons réalisé le diagramme de classes après avoir identifier toutes les classes du domaine.

Dans le chapitre suivant nous aborderons la phase de la conception dans la quelle nous présenterons notre module en toute bonne et due forme.

Chapitre V

Réalisation

« L'important n'est pas de suivre la ligne droite, mais de se rapprocher le plus du but. »

Tsar Lénine

Introduction

L'étape de réalisation est la dernière étape que nous allons aborder, elle vient après celle de la conception. La réalisation est une étape cruciale pour la mise en pratique de tout ce qu'on a fait auparavant.

Nous aborderons ce chapitre par la présentation et la définition des outils de réalisation à savoir le langage JAVA, l'IDE Eclipse, car il est impératif de connaitre les outils avec lesquels nous avons travaillé d'abord ensuite nous présenterons notre module en toute bonne et due forme.

1. Les outils de développement utilisés

Afin d'implémenté notre module nous avons utilisé quelques outils qui sont :

1.1. Langage de développement

Java est un langage de programmation orienté objet qui permet de développer des objets génériques de façon que le code puisse être réutilisable. Cette caractéristique nous a permis de développer des classes indépendamment du programme principal.

Notre choix pour ce langage est motivé par les critères suivants :

- Java nous a permis la programmation modulaire.
- Java est un langage indépendant de toute plate-forme d'exécution, ce qui signifie qu'un même programme peut fonctionner sur différentes plates-formes et sous différents systèmes d'exploitation. C'est l'un des principaux atouts de Java par rapport à d'autres langages de programmation.
- Java est doté d'une bibliothèque de classes très riche :
  - ✓ **Le graphisme :** cette bibliothèque est exploitée dans le développement des interfaces graphique (fenêtres et interactivité) relatives à son fonctionnement.
  - ✓ **La programmation concurrente *(multithreading)* :** cet avantage permis de synchroniser les différentes communications entre les composants lors du lancement de plusieurs processus au même temps.
  - ✓ **Gestion des exceptions :** c'est un mécanisme utile auquel nous avons fait appel pour gérer les erreurs inattendues lors de l'exécution de notre application. L'utilisation de ses bibliothèques facilite grandement la tache du programmeur lors de la construction des applications complexes. Pour la programmation, nous avons utilisé l'environnement de développent JDK *(Java*

Development Kit). JDK est un environnement dans lequel le code Java est compil´e pour être transformé en pseudo-code afin que la machine virtuelle de Java JVM (*Java Virtual Machine*) puisse l'interpréter. Il regroupe l'ensemble des composants permettant le développement et l'exécution des programmes Java **[20]**.

1.2. L'environnement de développement Eclipse IDE (Indigo)

Eclipse version *indigo* est un environnement de développement intégré **IDE** (*Integrated Development Environment*) dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. Les modules agissent sur des fichiers qui sont inclus dans l'espace de travail (*Workspace*). L'espace de travail regroupe les projets qui contiennent une arborescence de fichiers. Bien que développé en Java, les performances à l'exécution d'Eclipse sont très bonnes car il n'utilise pas *Swing* pour l'interface homme-machine mais un *toolkit* particulier nommé SWT associé à la bibliothèque JFace. SWT (*Standard Widget Toolkit*) est développé en Java par IBM en utilisant au maximum les composants natifs fournis par le système d'exploitation sous jacent. JFace utilise SWT et propose une API (*Application Programming Interface*) pour faciliter le développement d'interfaces graphiques.

1.2.1. Les points forts d'Eclipse

Eclipse possède de nombreux points forts qui sont à l'origine de son énorme succès dont les principaux sont :

- Une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de *plugins*.
- Un support multi langage grâce à des plugins dédiés : Cobol, C, PHP, etc.
- Malgré son écriture en Java, Eclipse est très rapide à l'exécution grâce à l'utilisation de la bibliothèque SWT.
- Un historique local des dernières modifications.
- La construction incrémentale des projets Java grâce à son propre compilateur qui permet en plus de compiler le code même avec des erreurs, de générer des messages d'erreurs personnalisés, de sélectionner la cible et de mettre en œuvre le *scrapbook* (permet des tests de code à la volée).
- Une exécution des applications dans une JVM dédiée sélectionnable avec possibilité d'utiliser un débogueur complet (points d'arrêts conditionnels, visualiser et modifier

des variables, évaluation d'expression dans le contexte d'exécution, changement du code à chaud avec l'utilisation d'une JVM).

- Une ergonomie entièrement configurable qui propose selon les activités à réaliser différentes "Perspectives" **[20]**.

2. Les classes principales de notre module

Parmi les classes principales de notre module, les classes de gestion de documents XML qui permettent de gérer ses documents (ajouter, modifier et supprimer), les classes de gestion de tables (ajouter, supprimer, modifier).

2.1. Classes gestion documents XML

2.1.1. Classe GestionXMLProtocol

Cette classe permet la gestion des documents XML, elle implémente l'interface **CDProtocol**, elle définit le comportement du protocole, parmi ces méthodes :

- **setXmlId(int id) :** cette méthode permet d'attribuer un identifiant à un document XML.
- **getXmlId() :** cette méthode permet de récupérer l'identifiant d'un document XML.
- **setXmlTaille(int id, int taille) :** cette méthode permet de donner une taille à un document XML spécifique.
- **getXmlTaille() :** cette méthode permet de retourner la taille d'un document XML.

2.1.2. Classe GestionXMLInitialise

Cette classe permet d'initialisée les documents XML, récupérer les données dans le fichier de configuration, elle est implémentée par l'interface **control**, la méthode principale est **execute()** dont laquelle sont déclarer les opérations d'initialisation.

2.1.3. Classe GestionXMLObserver

Cette classe implémente l'interface **control**, elle permet de récolter les différentes informations telle que nombre de nœud, le nombre de cycle et les détailles sur le document XML et de les afficher au cours de la simulation.

Dans la **figure 16** le fichier de configuration présente les configurations pour simuler l'ajout d'un document XML

```
# PEERSIM EXAMPLE : Ajouter un document XML

random.seed 1234567890
simulation.cycles 5
control.shf Shuffle
network.size 5

protocol.lnk IdleProtocol
protocol.avg example.gestion_donnees.GestionXMLProtocol
protocol.avg.linkable lnk


init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k 20

init.peak example.gestion_donnees.GestionXMLInitialise

init.peak.noeud 4          # noeud dans lequel on ajoute le document
init.peak.taille 10        # la taille du documet XML
init.peak.id 1             # Identifiant du document

init.peak.protocol avg


control.avgo example.gestion_donnees.GestionXMLObserver
control.avgo.protocol avg
```

Figure V.16 : Fichier de configuration « **Ajouter un document XML** ».

La figure ci-dessous (**figure V.17**), présente le résultat de la simulation pour le fichier de configuration de la **figure V.16** :

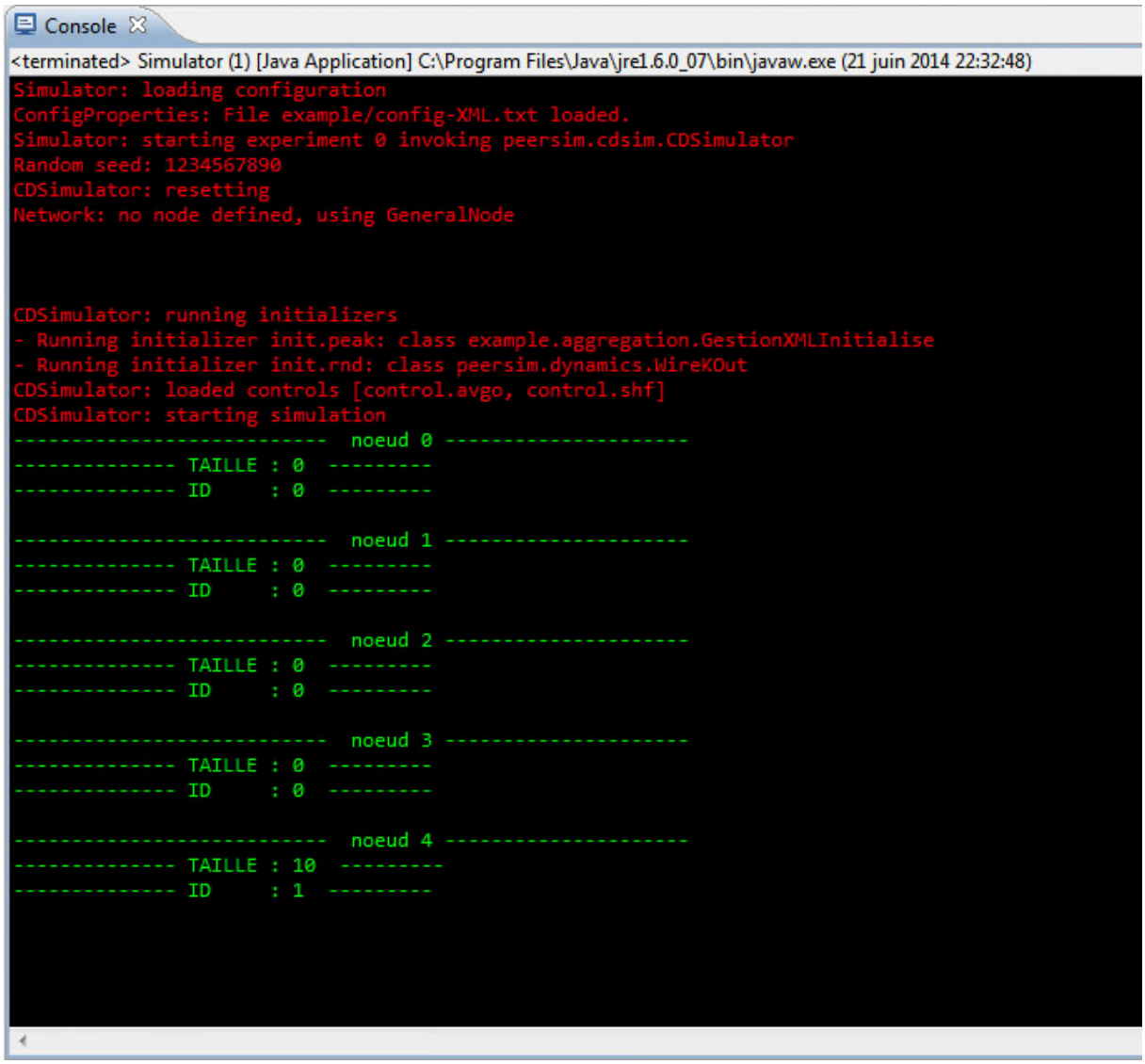

Figure V.17 : Résultat de la simulation « **Ajout d'un document XML** ».

Dans la **figure V.17** on voit d'abord le lancement des différents composants tel que l'initialisation du document XML, après c'est l'observateur qui va afficher le résultat de l'insertion du document dans le nœud 4.

2.2. Classes gestion table

2.2.1. Classe TableProtocol

Cette classe implémente l'interface **CDProtocol**, elle fournit les méthodes dont le rôle est de faire la gestion des tables, parmi ces méthodes :

- **getTailleDonnees() :** permet de retourner la taille de la table.
- **setTailleDonnees(int[] tableau, int taille) :** permet d'attribuer une taille à la table.
- **setDonnee(int[] tableau, int valeur, int position) :** permet d'insérer une valeur
- **setValueTuple(int[][] table, int tuple, int attribut, int valeur) :** cette méthode permet d'insérer une donnéedans un tuple.

2.2.2. Classe TableInitialise

Cette classe implémente l'interface **Control**, en exécutant la méthode **(execute())** elle permet d'initialisée les tables.

2.2.3. Classe TableObserver

Cette classe implémente l'interface **control**, elle permet de récolter les informations et de les afficher pendant la simulation.

Dans la **figure V.18** le fichier de configuration présente les configurations pour simuler l'ajout d'une table :

```
# PEERSIM EXAMPLE : ajouter une table

random.seed 1234567890
simulation.cycles 10
control.shf Shuffle
network.size 10                          #Proprieté globale de la simulation

protocol.lnk IdleProtocol
protocol.avg example.gestion_donnees.tableau  #Protocol de gestion de table
protocol.avg.linkable lnk

init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k 20


init.peak example.gestion_donnees.TableauInitializer  #Protocol qui initialise
                                                      #la table
init.peak.nbre_atr 4      #nombre d'attribut de la table


init.peak.at1 Nom
init.peak.at2 Prenom
init.peak.at3 Age                        #Nom d'attribut de la table
init.peak.at4 Moyenne

init.peak.attrib1 BOUAOUINA
init.peak.attrib2 Amirouche
init.peak.attrib3 26                     #tuple a ajouté
init.peak.attrib4 12


init.peak.protocol avg

control.avgo example.gestion_donnees.TableauObserver  #Classe qui affiche le
                                                          résultat
control.avgo.protocol avg
```

Figure 18 : Fichier de configuration « **Ajouter une table** ».

La figure ci-dessous (**figure 19**), présente le résultat de la simulation pour le fichier de configuration de la **figure 18** :

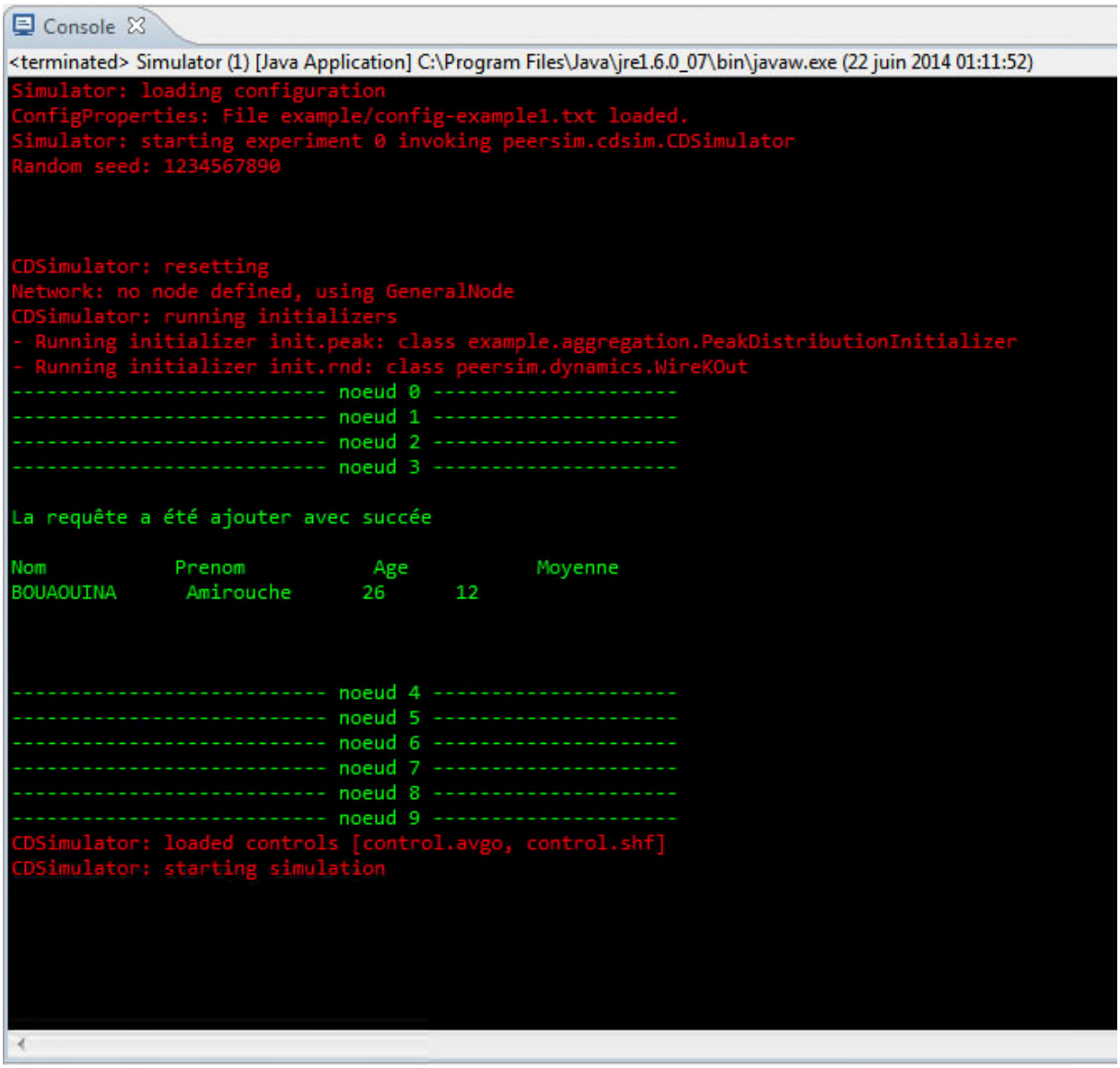

Figure 19 : Résultat de la simulation « **Ajout d'une table** ».

Dans la figure V.19 on voit d'abord le lancement des différents composants tel que l'initialisation de la tale, après c'est l'observateur qui va afficher le résultat de l'insertion du tuple dans le nœud 3.

Conclusion

Dans ce chapitre nous avons vu les différents protocoles que nous avons implémentés, à travers ces protocoles n'importe quel utilisateur pourra gérer les données (XML, Table, vues). Ce chapitre nous a permet d'enrichir nos connaissances en programmation JAVA.

Conclusion générale

Au cours de ce travail, nous avons vu les concepts sur les réseaux P2P et la gestion des données dans ce type de réseau, ainsi que le simulateur PeerSim qui fait l'objet de notre projet. Cette partie théorique à pour but d'enrichir nos connaissances pour mieux aborder la partie pratique de notre projet.

Afin d'aboutir à la satisfaction des besoins des utilisateurs (utilisateur qui simule avec PeerSim), nous avons entrepris la conception en utilisant le formalisme UML qui est un langage de modélisation, il est bien pour structurer, représenter et bien organiser nos idées. À travers UML nous avons fait recours au processus 2TUP qui nous a facilité la tache.

Pour notre application nous avons choisis d'utiliser l'IDE Eclipse, ce choix est du a sa facilité d'utilisation et à sa maniabilité et surtout pour son efficacité, même si nous avons eu à faire à quelques problèmes et à quelques bugs de temps en temps, mais il reste un très bon logiciel et un outil complet pour tout développeur.

Avec toutes ces méthodes, techniques et outils nous avons pu finir notre conception et réaliser notre module, malgré l'insuffisance de temps imparti et les quelques petits obstacles organisationnels rencontrés durant ce travail.

Ce projet nous a permis d'enrichir nos connaissances théoriques et nos modestes compétences dans le domaine de la conception et de la programmation orienté objet. Comme il nous a autorisé la mise en application des connaissances acquises tout au long des études universitaire en nous mettant dans un environnement de pratique.

Le travail maintenant terminé, il ne nous reste qu'à souhaiter que notre module répond bien aux exigences des développeurs, et aussi que le module conçu c'est-à-dire le module de gestion de données pour PeerSim les aidera vraiment à bien Simuler leurs programmes et protocoles.

BIBLIOGRAPHIE

[1] Patrick MARLIER, sécurité du peer-to-peer, labo-asso, Juin 2006

[2] Anne BENOIT, Algorithmique des réseaux et des télécoms, cours (ENS Lyon, M1),2006

[3] Nathalie BUDAN ; Benoit TEDESCHI, Stéphane VAUBOURG, Nouvelles Technologies Réseau.

[4] Dominique POURE ; Dimitri CARON, l'empire du «peer to peer», université DAUPHINE, février 2007.

[5] Stephanos Androutsellis-Theotokis ; Diomidis Spinellis, A Survey of Peer-to-Peer Content Distribution Technologies, ACM Computing Surveys, Vol. 36, No. 4, Decembre 2004.

[6] Patrick MARLIER, Sécurité du Peer-To-Peer, 2006.

[7] Bruno DEFUDE, P2P simulation with PeerSim, January/February 2007.

[8] Benoît MERLET ; Pierre ZUREK, Notre implémentation du simulateur, 16 Aout 2005.

[9] M. Raddad, Thèse en vue de l'obtention du doctorat, Localisation de sources de données et optimisation de requêtes réparties en environnement pair-à-pair, Université de Toulouse, 11 mai 2010.

[10] Intégration de sources de données á base ontologique dans un environnement P2P.pdf

[11] Jin LI, Multimedia 06 proceedings of the 14th annual acm international, conference on multimedia, 2006.

[12] Asterios KATSIFODIMOS**,** Techniques efficaces basées sur des vues matérialisées pour la gestion des données du Web : algorithmes et systèmes, université PARIS-SUD, 10/2013

[13] http://georges.gardarin.free.fr/Livre_BD_Contenu/9-Vue.pdf.

[14] http://msdn.microsoft.com/en-us/library/ms151196.aspx.

[15] Pascal ROQUES et franck VALLEE, UML2 en action, De l'analyse des besoins à la conception, 4ème édition.

[16]T.AMGHAR **,**F.YEMMLI**,**K.SACI **,**F.ABDELFETTAH**,**S.TOULA**,**M.AHMEDKHODJA,D. CHEROUFA,K. AOUDIA,T. HAMLAT,N. HAMOUCHE**,** Conception et réalisation d'un site web d'une agence immobilière, Université A. Mira Bejaia, 2009/2010.

[17] UML2 analyse et conception, édition Dunod, Paris, 2008.

[18] UML2 analyse et conception, édition Dunod, Paris, 2008.

[19] Christian SOUTOU, UML 2 pour les bases de données, édition Eyrolles.

[20] Michel DIVAY. *La programmation objet en java*, EYROLLES, 2006.

Résumé

Au cours des dernières années, Les technologies Peer-to-Peer(P2P) sont devenues de plus en plus populaire. Le développement de système de P2P a beaucoup de problèmes communs trouvés avec la programmation d'autres systèmes répartis. Les simulateurs de réseau de P2P essayent de résoudre la difficulté accrue en concevant ou en examinant des systèmes de P2P, en fournissant un environnement virtuel de réseau. Les simulateurs de réseau de P2P sont généralement utilisés pour simuler les communications réseau dans des scénarios ou situations particulières, sans avoir à configurer les machines ou les réseaux réels. Ils existent plusieurs simulateurs; **PeerSim** est l'un des plus connue et fait l'objet de notre projet. Il propose un grand nombre de ses modules dans ses sources, qui facilitent grandement le codage de nouvelles applications. Malgré ça, **PeerSim** ne permet pas aux utilisateurs de gérer les données, créer et initialiser des requêtes. L'objectif visé par ce projet, est de concevoir et implémenter un module de gestion de données pour **PeerSim**.

Pour développer ce module, Nous avons utilisé le formalisme graphique proposé par UML, Ce dernier propose des diagrammes qui ont permis de spécifier les besoins du projet, de les analyser, et enfin de décrire l'architecture générale du système. Pour la conception UML est soutenu par 2TUP qui est un processus de développement logiciel qui implémente le Processus Unifié. Pour ce qui est de la réalisation et l'implémentation de notre module, notre choix s'est porté sur le langage JAVA et l'IDE Eclipse puisque la combinaison de ces deux nous offre un résultat appréciable et répond au mieux à nos attentes.

Mot clé : P2P, PeerSim, JAVA, Eclipse.

Abstract

In recent years, technologies Peer-to-Peer (P2P) have become increasingly popular. The development of P2P system has many common problems found with other programming distributed systems. The P2P network simulators attempt to solve the increasing difficulty in designing or examining P2P systems, providing a virtual network environment. The P2P network simulators are generally used to simulate network communications in specific scenarios or situations, without having to configure machines or real networks. They exist several simulators; PeerSim is one of the most known and is the subject of our project. It offers a large number of modules in its sources, which greatly facilitate the encoding of new applications. Despite this, PeerSim does not allow users to manage data, create and initialize requests. The objective of this project is to design and implement a data management module forPeerSim

To develop this module, we used the proposed UML graphical formalism latter offers diagrams that helped specify the project requirements, analyze, and finally describe the overall architecture of the system. To design UML is supported by 2TUP which is a software development process that implements the Unified Process. Regarding the realization and implementation of our module, our choice fell on the Java language and Eclipse IDE since the combination of these two gives us a significant result and best meets our expectations.