UNIVERSITÉ DU QUÉBEC À MONTRÉAL

NETSIM :
UN LOGICIEL DE MODÉLISATION ET DE SIMULATION DE RÉSEAUX D'INFORMATION

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
MÉLANIE LORD

AVRIL 2007

REMERCIEMENTS

Ce document témoigne de l'achèvement d'un long processus de recherche que j'ai entrepris il y a deux ans déjà. Au cours de cette aventure, la chance m'a été donnée de côtoyer nombre de gens qui ont eu, directement ou indirectement, une influence positive sur l'accomplissement de ce projet. Ils ont été mes formateurs, mes amis, mes conseillers, mes collègues, ma famille et chacun d'eux, à leur manière, a contribué à la réalisation de ce mémoire.

Je souhaite remercier M. Daniel Memmi, professeur au département d'informatique à l'Université du Québec à Montréal, de m'avoir dirigée dans mes travaux de recherche. Il a su me guider et me donner les outils et les ressources nécessaires pour mener à terme ce projet.

Je tiens aussi à remercier Alexandre Blondin-Massé, étudiant à la maîtrise en informatique à l'Université du Québec à Montréal, pour sa gentillesse et le temps qu'il a consacré à la programmation de certaines composantes du logiciel NetSim.

J'ai effectué mon travail de recherche, en grande partie, au laboratoire d'informatique LATECE. Je remercie les étudiants de ce laboratoire de constamment contribuer à l'ambiance stimulante qui y règne. Plus particulièrement, je remercie Ghizlane El Boussaidi, étudiante au doctorat en informatique à l'Université de Montréal, pour ses bons conseils.

Enfin, je remercie ma famille dont le soutien est toujours inconditionnel.

Table des matières

Liste des figures

Liste des tableaux

Résumé

Les réseaux électroniques basés sur Internet ont beaucoup accéléré la circulation de l'information dans notre société moderne, mais on commence à voir que les échanges d'information s'effectuent d'abord dans le cadre de réseaux sociaux. Le Web comporte également une structure en réseau, mais celle-ci est assez particulière.

Ces différents types de réseaux d'information montrent à la fois des caractéristiques communes et des spécificités dont il convient de tenir compte. Pour des raisons à la fois techniques, sociales et économiques, il est donc utile de chercher à modéliser les réseaux par lesquels circulent information et connaissances.

En nous inspirant des acquis importants en sociologie structurale et en analyse mathématique de réseaux, nous avons développé une approche de modélisation des réseaux par simulation. Nous avons tout d'abord développé un langage de modélisation qui se veut le plus flexible possible tout en demeurant simple et abordable pour des utilisateurs variés. Pour ce faire, nous nous sommes inspirés de modèles existants de la littérature et avons tenté d'en abstraire les concepts essentiels que devrait offrir un tel langage.

Ensuite, nous avons réalisé NetSim, une plate-forme paramétrable de génération de réseaux capable d'interpréter ce langage et permettant de tester diverses hypothèses sur la structure des réseaux que l'on peut observer empiriquement. Cet outil offre aussi des fonctionnalités de visualisation, sous forme de graphe, de l'évolution du réseau dans le temps.

Finalement, nous avons obtenu des résultats plutôt convaincants quant à l'utilisation de notre logiciel pour modéliser et simuler divers types de modèles de l'évolution temporelle de la structure des réseaux. Les phénomènes observés par simulation s'apparentent effectivement à certains faits observés dans la réalité.

Ce logiciel pourra servir d'outil de recherche, d'expérimentation, de visualisation et de formation dans un domaine en plein développement.

Mots-clés : Réseaux d'information, réseaux sociaux, modélisation, simulation.

CHAPITRE 1

Introduction

1.1 Contexte

Les réseaux d'information sont des *réseaux par lesquels circulent information et connaissances*. Cette définition est assez vague, mais exclut tout de même un bon nombre de réseaux : les réseaux de transport comme les réseaux routiers ou les lignes aériennes, les réseaux de distribution comme les aqueducs, le système sanguin, les systèmes de livraisons par camions ou par avions, etc. Cependant, cette distinction n'est pas toujours aussi claire. Par exemple, les réseaux routiers servent aussi à distribuer le courrier qui est en soi de l'information.

Ce que nous entendons par information et/ou connaissances consiste en tout ce qui est utile et pertinent à une entité cognitive pour accomplir une tâche, résoudre un problème, comprendre une situation, faire des choix, planifier ou adopter tel ou tel comportement dans des situations diverses. Par exemple, si je sais, par le biais de la météo, qu'il va pleuvoir le lendemain, je ne planifierai certainement pas d'aller faire un pique-nique. Nous insistons ici sur le caractère de la pertinence. Une information inutile n'est pas vraiment une information au sens où nous l'entendons puisqu'elle ne détermine aucunement nos comportements ni ne motive nos choix. En ce sens, le degré de pertinence d'une information dépend d'un contexte. En effet, le fait de savoir s'il pleuvra le lendemain m'importe peu si je planifie de rester à la maison. Si je suis pilote d'avion, par contre, ma connaissance des conditions météorologiques aura beaucoup plus d'importance que si je travaille comme bibliothécaire.

Au cours d'une journée, un professionnel moderne utilise des milliers d'informations pour accomplir son travail. Ces informations proviennent de plusieurs sources telles que les livres

spécialisés, les graphiques, les cartes, les encyclopédies, les codes et conventions, les ordinateurs, le courrier électronique, le téléphone, les discussions entre collègues, les fichiers et les bases de données, le WWW, etc.

Nous utilisons le concept de réseaux d'information pour tenter de décrire la manière dont l'information et les connaissances sont partagées entre diverses entités; pour essayer de formaliser ces échanges d'information.

Les réseaux par lesquels circulent information et connaissances sont nombreux. Notre corps, par exemple, contient plusieurs réseaux qui permettent à l'information de circuler afin que celui-ci puisse fonctionner normalement. Parmi ces réseaux, on peut mentionner le système nerveux formé de neurones qui communiquent entre eux par la transmission de messages chimiques appelés neurotransmetteurs. Si ce système de communication se détériore, le fonctionnement normal du corps est en péril. Par exemple, la maladie de Parkinson, qui est une affection dégénérative du système nerveux, est causée par le manque de production de dopamine (neurotransmetteur) dans le système nerveux. Cela entrave la bonne transmission des messages entre les neurones et il en résulte une rigidité musculaire et des tremblements du corps.

Le Web est aussi une vaste source d'information. Sa structure, formée de pages Web étant reliées entre elles par des hyperliens, forme un immense réseau passif, mais dans lequel on peut naviguer à l'aide d'un fureteur Web. On y transmet des requêtes (URL) et au retour, on reçoit des pages d'information.

Nous remarquons cependant que, parce que nous fonctionnons la plupart du temps en société, la majorité des informations que nous recevons proviennent de notre milieu social et sont pertinentes au sein même de ce milieu. Nous parlons ici d'un type particulier de réseaux d'information : les réseaux sociaux. Le concept de réseaux sociaux tente de formaliser plus particulièrement les échanges d'information entre des individus ou des groupes sociaux. Ces échanges s'effectuent par les relations qu'entretiennent les entités sociales entre elles. Nous échangeons constamment de l'information et des connaissances avec nos amis ou nos

collègues de travail. Nous entretenons différents types de relations : relations filiales, d'amitié, de conseil, de pouvoir, de collaboration, d'aide, de coopération, etc. La nature de l'information que nous échangeons avec une personne dépend du type de relation que nous entretenons avec celle-ci. En effet, nos sujets de discussion seront certainement différents selon que nous discutons avec un ami proche, un collègue de travail ou notre médecin de famille. Bref, les réseaux sociaux sont des réseaux par lesquels circule un bon nombre d'informations et de connaissances qui nous sont très utiles dans la gestion de nos comportements sociaux.

L'évolution des technologies a grandement favorisé les échanges sociaux. Les réseaux téléphoniques, par exemple, permettent la transmission d'informations entre individus. Avec l'arrivée de l'Internet, les réseaux se sont multipliés : outre le courrier électronique, on assiste à l'émergence de nombreuses communautés virtuelles dont les membres communiquent entre eux en utilisant des outils comme les blogues, les wikis, les forums et les espaces virtuels de collaboration. Le WWW n'est pas un réseau social en soi, mais peut s'y apparenter en ce sens que les pages Web (information) sont conçues par des individus et peuvent être consultées par des individus. Contrairement aux réseaux sociaux, le Web est une structure passive, mais qui permet, d'une certaine manière, l'échange d'informations entre individus.

1.2 Problématique

Plus on a accès à l'information et aux connaissances, meilleures sont nos chances de réussite dans l'accomplissement de nos buts. Cela est vrai dans tous les domaines. Que ce soit pour prédire la tendance du marché en économie, pour rédiger un article scientifique, pour obtenir une promotion, pour accroître notre pouvoir de négociation, pour gérer le personnel d'une entreprise, pour demeurer compétitif sur le marché, pour décider d'investir ou non dans une relation, l'information est une ressource indispensable.

Ainsi, pour des raisons à la fois techniques, sociales et économiques, il est donc utile de chercher à comprendre les réseaux par lesquels circulent information et connaissances. Il existe déjà un certain nombre d'approches pertinentes.

Pour les réseaux sociaux en particulier, la sociologie structurale a élaboré des méthodes formelles d'analyse de réseaux basées sur la théorie des graphes (Lazega, 1998). Cependant, ces enquêtes sociologiques restent le plus souvent lentes et laborieuses, car elles mobilisent un grand nombre de ressources : elles sont souvent très coûteuses, demandent la participation d'un grand nombre d'individus et peuvent parfois s'échelonner sur plusieurs années. De plus, les études empiriques concernant l'évolution temporelle de la structure des réseaux sociaux sont essentielles, mais restent encore à un état embryonnaire à cause du manque de données de réseaux propres à ce genre d'étude. En effet, il n'est pas facile d'obtenir des données de réseaux pertinentes s'échelonnant sur une bonne période de temps et présentant toute l'information nécessaire à la reconstruction d'un réseau du passé et de son évolution temporelle. Bien que ces études soient assez rares, elles ne sont pas inexistantes. Je citerai comme exemple l'étude des réseaux de collaboration scientifique, en physique et en biologie, conduite par (Newman, 2001), pour tester l'hypothèse que la transitivité ainsi que l'attachement préférentiel sont des propriétés de certains réseaux sociaux.

Ce réseau de collaboration scientifique comprend des individus qui sont reliés ensemble s'ils sont co-auteurs d'un même ouvrage scientifique. Pour ce faire, Newman tira les informations de deux sources bibliographiques :

1. The Los Alamos E-print Archive, une base de données en physique dont les ouvrages sont soumis par les auteurs eux-mêmes.

2. Medline, une base de données d'ouvrages publiés en biologie et en médecine qui est professionnellement maintenue par les instituts nationaux de la santé (National Institutes of Health).

Bien qu'aucune de ces bases de données n'enregistre la date de publication exacte des ouvrages qu'elles contiennent, les deux peuvent fournir l'ordre dans lequel les ouvrages ont été ajoutés à leur base de données respective.

Newman a donc recueilli les données de collaboration sur un intervalle de six années en prenant les données des cinq premières années pour reconstruire le réseau de collaboration. Il

a ensuite étudié l'évolution de la topologie du réseau en considérant le passage de la structure du réseau obtenu les cinq premières années à la structure du réseau obtenu à la sixième année.

Pour ce qui est des réseaux électroniques basés sur Internet, il est possible de les étudier en recueillant des traces électroniques de communications entre individus et de reproduire le graphe de ces échanges à des fins d'analyse. Nous citerons comme exemple l'article de Paul Mutton (Mutton, 2004) dans lequel celui-ci décrit une méthode pour inférer le réseau social à partir d'un groupe d'utilisateurs d'IRC (Internet Relay Chat). Tout d'abord, un robot (bot) IRC est utilisé pour écouter sur un canal donné. Celui-ci procède ensuite à une analyse heuristique des événements recensés et crée une approximation mathématique du réseau social. Finalement, le robot peut produire une visualisation graphique du réseau inféré. On obtient alors un réseau d'interactions qui peut être analysé.

Les courriels sont aussi des traces électroniques de communications entre personnes qui peuvent être utilisées pour recréer le réseau d'interactions. Par exemple, (Viegas et al., 2004) décrivent deux logiciels qui utilisent les archives de courriels d'un individu pour recréer son réseau social personnel. Ces logiciels analysent les réseaux sociaux qui peuvent être dérivés des informations concernant l'envoyeur, le ou les destinataires, le sujet et la date d'envoi des courriels. En analysant les destinataires des courriels, il est possible de dériver un graphe des relations sociales pouvant ensuite être visualisé. De plus, selon la date des courriels, on peut recréer et visualiser l'évolution de ce réseau social dans le temps.

L'étude de la structure du Web est aussi pertinente pour comprendre les phénomènes sociologiques qui caractérisent son évolution. Considérons le Web comme un graphe dirigé dans lequel les pages Web sont les nœuds et les hyperliens sont les arcs. On peut alors se demander, par exemple, quel est son diamètre, à quoi ressemble la distribution des degrés des noeuds ou bien quelle est la longueur moyenne du chemin allant d'une page Web à n'importe quelle autre (ces notions seront explicitées par la suite). La compréhension de la structure du Web permet, par exemple, d'améliorer les algorithmes d'exploration de la Toile et de recherche d'informations.

Pour étudier et analyser la structure du Web, il faut d'abord la cartographier. Pour ce faire, on peut commencer avec quelques pages Web et découvrir d'autres pages en suivant les hyperliens des pages initiales. Puis, on suit les liens des pages nouvellement découvertes pour en découvrir d'autres et ainsi de suite. On construit ainsi un graphe qui représente la topologie du Web. Cependant, cette technique a ses limitations. Il est en effet impossible de couvrir le Web dans son entier, car si aucune des pages Web découvertes au cours du processus de cartographie ne pointe vers une page particulière, cette dernière ne sera jamais trouvée. De plus, les pages Web sont tellement nombreuses (quelques milliards) que cela rend le processus d'exploration très long. La cartographie du Web ne peut donc être que partielle. Un exemple d'étude du Web par cartographie est donné dans l'article de (Broder et al, 2000).

En résumé, les enquêtes sociologiques sur le terrain mobilisent un grand nombre de ressources. De plus, la collecte des données est un problème difficile lorsqu'il s'agit d'étudier l'évolution temporelle de la structure de réseaux d'information divers. Finalement, dans le cas particulier de la cartographie du Web, celle-ci est assez coûteuse et ne peut être que partielle étant donné l'immensité de ce réseau.

Bien que ces méthodes demeurent essentielles, la modélisation et la simulation par ordinateur nous semblent indiquées, dans certains cas, pour pallier ce problème. Un modèle n'a cependant pas la prétention de représenter parfaitement la réalité. Dans cette optique, la modélisation ne doit pas être vue comme une alternative à l'étude de terrain, mais plutôt comme une approche complémentaire très utile pour tester ou formuler certaines hypothèses. Les résultats de simulations devront tôt ou tard être comparés aux phénomènes observés dans la réalité, mais nous croyons la modélisation présente tout de même un certain nombre d'avantages pour la compréhension de phénomènes particuliers. Elle demeure un outil à ne pas négliger tout en gardant à l'esprit les limites de nos modèles.

Il existe déjà plusieurs modèles de réseaux d'information provenant de diverses disciplines telles que la physique, les mathématiques et la sociologie. Ces modèles tentent de représenter partiellement des phénomènes particuliers observés dans le monde réel, tels que la

transitivité, l'effet des petits mondes et l'homophilie dans les réseaux sociaux ou l'attachement préférentiel dans le réseau du Web.

Ces modèles sont pour la plupart des modèles statiques, mais de plus en plus, on commence à concevoir des modèles dynamiques de l'évolution de la structure de réseaux d'information dans le temps (Jin, Girvan et Newman, 2001 ; Albert et Barabási, 2000). Les modèles statiques sont souvent définis par un algorithme qui permet de construire une structure particulière de réseau qui possède les caractéristiques que l'on veut démontrer. Un modèle dynamique, quant à lui, fait intervenir la notion de temps. Il consiste en un ensemble de règles servant à construire ou modifier le réseau. En général, ces règles doivent être effectuées, selon certaines probabilités ou conditions, à chaque pas de temps. On peut alors observer une modification de la topologie du réseau dans le temps. Le but de ce type de modélisation est souvent d'étudier les mécanismes qui produisent des structures de réseaux possédant certaines caractéristiques observées dans le monde réel.

Une grande partie des modèles existants prennent la forme d'un graphe dans lequel les sommets représentent les entités du réseau et les arcs ou arêtes du graphe représentent les relations entre ces entités. Par exemple, dans le cas du Web, les sommets représentent les pages Web et les arcs représentent les hyperliens qui relient les pages entre elles. Pour un réseau social, les sommets représentent les individus (acteurs) de l'ensemble social et les liens représentent les relations (d'amitié, par exemple) qu'ils ont entre eux. La représentation sous forme de graphe est particulièrement intéressante, car elle permet l'utilisation de la théorie des graphes et de l'algèbre matricielle pour l'analyse de la structure des réseaux.

La visualisation du réseau sous forme de graphe est aussi très avantageuse. Elle permet de constater rapidement, sans autre analyse, s'il semble se former certains agencements particuliers dans la structure du réseau étudié. La figure 1.2.1 illustre quelques exemples de structures particulières pouvant être facilement repérées par la visualisation graphique.

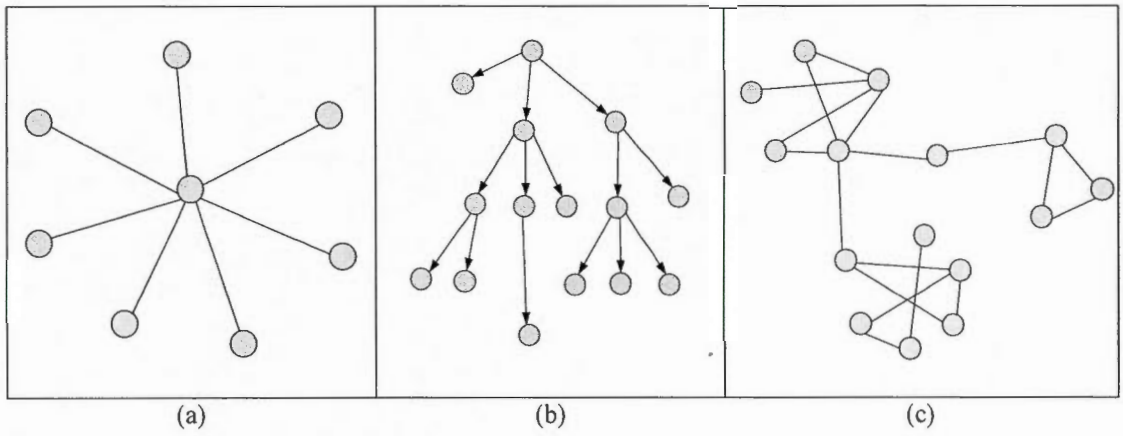


Figure 1.2.1 Repérage de structures typiques par visualisation graphique
La visualisation graphique est un atout considérable dans l'étude structurale des réseaux. Elle permet de repérer rapidement certaines caractéristiques du réseau. Supposons, par exemple, que les réseaux ci-dessus sont des réseaux sociaux. La figure (a) montre une structure en forme d'étoile. On y remarque facilement un acteur central qui de ce fait, a beaucoup d'influence sur la circulation de l'information vers les acteurs périphériques, car chaque acteur périphérique doit passer par l'acteur central pour communiquer avec les autres. La figure (b) montre une structure en forme d'arborescence qui induit tout de suite l'idée d'une hiérarchie. La figure (c) montre un réseau où l'on remarque facilement la création de petites communautés distinctes au sein du réseau.

Dans le cadre de ce travail, les modèles qui nous intéressent plus particulièrement sont les modèles dynamiques pouvant être représentés sous forme de graphe. Étant donné qu'ils sont dynamiques, nous pouvons les simuler par ordinateur pour observer leurs changements structuraux au cours du temps. Bien qu'il existe beaucoup d'outils (logiciels et librairies) qui aident à la programmation d'un modèle dans le but de le simuler et de le visualiser, on constate que la plupart de ces outils sont assez complexes et demandent souvent une bonne connaissance de divers langages de programmation. Aussi, dans bien des cas, la programmation de la simulation est propre à un modèle ou une classe de modèles en particulier et tout le travail de programmation est à recommencer lorsque l'on veut simuler un modèle différent.

Dans cette optique, nous avons élaboré un langage de modélisation de réseaux d'information simple et abordable pour des utilisateurs variés. Nous avons ensuite réalisé NetSim, une plate-forme de simulation paramétrable, capable d'interpréter ce langage de modélisation. NetSim permet de modifier facilement et rapidement les règles et les paramètres de divers

modèles et permet aussi la visualisation, sous forme de graphe, de l'évolution de la structure du réseau dans le temps.

1.3 Objectifs de recherche et méthodologie

1.3.1 Création d'un langage de modélisation de réseaux d'information

Notre premier objectif est la création d'un langage de modélisation de réseaux d'information facile à utiliser pour les concepteurs de modèles qui, le plus souvent, ne sont pas des informaticiens. Notre but est de trouver une façon d'exprimer les règles des modèles dynamiques de manière intuitive pour l'utilisateur, qui reflète l'esprit du modèle. Pour ce faire, nous avons observé différents modèles dynamiques de réseaux existants (Jin, Girvan et Newman, 2001 ; Barabási et Albert, 1999 ; Albert et Barabási, 2000) et avons tenté d'en abstraire les concepts essentiels que devrait exprimer un tel langage. En particulier, nous avons passé en revue les entités, les structures et les actions de base nécessaires pour modéliser ce domaine. Ce langage a d'abord été élaboré pour décrire des modèles de l'évolution de la structure des réseaux sociaux et du Web, mais reste assez souple pour exprimer une certaine variété d'autres modèles de réseaux d'information.

Parce que nous désirons concevoir un langage qui, d'une part, est aussi générique (flexible) que possible, mais d'autre part, demeure simple et abordable pour des utilisateurs divers, nous avons dû cerner le champ d'application de notre langage de modélisation par quelques principes généraux. Les principes suivants aident à déterminer les types de modèles dynamiques pouvant être traduits dans ce langage :

1. Le modèle doit représenter le réseau d'information sous forme de graphe.

2. L'ensemble des règles du modèle s'applique, selon certaines probabilités ou conditions, à chaque pas de temps.

3. Le modèle peut définir des attributs sur les nœuds, les liens ou le réseau global pourvu que ceux-ci soient numériques. L'ensemble des valeurs de chaque attribut au temps t est ce qui détermine l'état du réseau au temps t. Les valeurs des attributs peuvent changer au cours du temps, selon certaines probabilités ou conditions qui peuvent dépendre de l'état du système.

4. Les règles du modèle peuvent concerner la suppression ou l'ajout de nœuds ou de liens ou bien la modification des attributs des nœuds, des liens ou du réseau. Le modèle peut spécifier l'exécution des règles en fonction de l'état du système ou selon certaines probabilités ou conditions qui peuvent dépendre de l'état du système.

1.3.2 Implémentation d'un logiciel d'interprétation du langage de modélisation

Notre deuxième objectif est de concevoir un logiciel de simulation (NetSim) capable d'interpréter les modèles écrits dans notre langage de modélisation et offrant des fonctionnalités de visualisation de l'évolution temporelle de la structure des réseaux simulés.

Nous voulons plus particulièrement concevoir une plate-forme générique de génération de réseaux qui peut être facilement extensible. Nous avons donc déterminé quelques fonctionnalités importantes à réaliser dans le cadre de ce travail, mais nous voulons concevoir ce logiciel dans l'optique de faciliter l'implémentation éventuelle de nouvelles fonctionnalités.

L'aspect visualisation, sous forme de graphe, est aussi une caractéristique importante que nous désirons implémenter. Comme nous l'avons vu brièvement à la section 1.2, la visualisation graphique permet de reconnaître instantanément certains motifs structuraux dans le réseau simulé. Dans certains cas, cette reconnaissance instantanée peut rapidement valider ou invalider nos hypothèses initiales ou alors nous conduire à formuler d'autres hypothèses sur les caractéristiques de la structure des réseaux étudiés.

Il s'agit à la fois d'étudier des phénomènes sociaux remarquables et de contribuer à la modélisation de réseaux variés. La plupart des travaux techniques actuels ne tiennent pas suffisamment compte des aspects sociaux de la communication et restent ainsi en deçà de leur potentiel d'utilisation. Nous attendons de ce projet la possibilité de modéliser finement des réseaux pertinents pour la circulation de l'information. Cela nous semble très utile, par exemple, pour le développement de nouvelles applications de communication électronique ou de travail collaboratif par ordinateur.

Ce logiciel pourra servir d'outil de recherche, d'expérimentation, de visualisation et de formation dans un domaine en plein développement. Le système est conçu en ce but, avec une interface appropriée pour contribuer en particulier à la modélisation de l'évolution temporelle des réseaux.

1.3.3 Simulation de réseaux d'information divers à l'aide de NetSim

Notre dernier objectif de recherche est de tester différents modèles dynamiques de réseaux d'information existants à l'aide de NetSim. Nous voulons alors comparer les résultats obtenus de nos simulations avec les résultats de simulation obtenus par les auteurs des modèles étudiés et avec les phénomènes observés dans le monde réel.

1.4 Plan du mémoire

Dans le chapitre 2, nous expliquons différentes notions qui concernent un type particulier de réseaux d'information : les réseaux sociaux. Ce chapitre vise principalement à présenter le contexte théorique qui a motivé la conception d'une plate-forme générique de modélisation et de simulation de réseaux d'information et plus particulièrement, de réseaux sociaux.

Dans un premier temps, nous montrerons que les réseaux sociaux occupent une grande place dans notre société et qu'il est pertinent de vouloir chercher à les comprendre. En second lieu, nous expliquerons quelques méthodes d'analyses des réseaux sociaux tirées de la sociologie structurale. Ensuite, nous verrons que l'étude des réseaux sociaux, par modélisation, a donné

lieu à une grande variété de modèles provenant de domaines divers et nous commenterons quelques-uns de ces modèles. Finalement, nous discuterons des types d'outils qui existent pour modéliser, simuler et visualiser les réseaux sociaux en les positionnant par rapport au logiciel NetSim.

Au chapitre 3, nous présentons le langage de modélisation de NetSim. Nous verrons les entités, les attributs et les fonctions qui le composent et la façon d'exprimer, dans ce langage, les définitions et les règles des modèles que l'on veut simuler. Pour ce faire, nous commenterons plusieurs exemples pour finalement écrire un modèle complet tiré de la littérature.

Dans le chapitre 4, nous expliquons l'architecture et le fonctionnement de NetSim. Nous présenterons tout d'abord une vue d'ensemble du fonctionnement de cette plate-forme pour ensuite expliquer, de façon plus détaillée, certaines fonctionnalités que nous considérons comme les plus importantes.

Le chapitre 5 présente les résultats obtenus des simulations que nous avons effectuées avec NetSim. Nous modéliserons différents modèles de la littérature à l'aide du langage de modélisation de NetSim et comparerons nos résultats de simulation avec les résultats obtenus par les auteurs de ces modèles. Aussi, nous tenterons de faire le parallèle entre nos résultats et les phénomènes que nous pouvons observer dans la réalité.

Finalement, dans le chapitre 6, nous présentons la conclusion de ce travail de recherche. Nous rappellerons nos principales réalisations et proposerons différentes avenues de recherche qui seraient intéressantes pour la continuation de ce projet dans le futur.

CHAPITRE 2

Les réseaux sociaux

2.1 Introduction

Nous consacrons un chapitre entier sur les réseaux sociaux, car ce type particulier de réseaux d'information est un domaine de recherche en plein développement tout en ayant une assez longue histoire. En effet, les réseaux électroniques basés sur Internet ont beaucoup accéléré la circulation de l'information dans notre société moderne, mais on commence à voir que les échanges d'informations s'effectuent d'abord dans le cadre de réseaux sociaux.

On remarque, entre autres, l'apparition massive de logiciels dits sociaux (ex. : les wikis et les blogues). Les outils de collaboration traditionnels (collecticiels), orientés projet, avaient pour but d'aider les organisations à communiquer, collaborer et à coordonner leurs activités afin d'accomplir une tâche commune aux membres du groupe (Ellis, Gibbs et Rein, 1991). Les logiciels sociaux visent un tout autre objectif. Ces logiciels sont destinés à promouvoir les interactions entre personnes incluant la communication en temps réel (ex. : messagerie instantanée) ou en temps différé (ex. : espace virtuel de collaboration, messagerie électronique, forums). Ils offrent aussi la possibilité aux individus ou aux groupes d'évaluer les contributions de leurs collègues et supportent la représentation et la gestion électronique des réseaux sociaux personnels des individus tout en leur permettant de les développer (Lamy, 2004).

Dans ce chapitre, nous voyons comment le développement remarquable des réseaux sociaux influence, entre autres, le domaine de la gestion de la connaissance, le développement d'outils pour l'aide à la recherche d'informations, la tendance, de plus en plus grande, qu'ont

les entreprises à se regrouper en réseaux pour affronter le marché ainsi que la perception de ce qui pourrait bien être la future génération de l'Internet.

En second lieu, nous présentons quelques méthodes d'analyse de réseaux sociaux proposés par la sociologie structurale, car celles-ci peuvent s'avérer très utiles pour comprendre et analyser la structure des réseaux générés par modélisation et simulation.

Ensuite, étant donné que nous nous intéressons particulièrement à l'étude des réseaux sociaux par modélisation, nous présentons une revue partielle des modèles actuels. Encore une fois, on remarque que l'intérêt pour les réseaux sociaux se retrouve dans plusieurs disciplines. En effet, nous verrons des modèles provenant de champs d'études aussi variés que la physique, les mathématiques, l'économie et la sociologie.

Finalement, nous positionnons les principaux outils de modélisation et de simulation de réseaux sociaux par rapport au logiciel que nous proposons dans ce projet.

2.2 Un domaine de recherche en pleine expansion

2.2.1 Les réseaux sociaux dans la gestion de la connaissance

La gestion de la connaissance (knowledge management) s'intéresse à l'organisation, la création, le partage et la circulation de la connaissance au sein d'une organisation : elle cherche à mettre à profit la connaissance déjà disponible et tente de susciter la création de nouvelles connaissances.

La première génération de la gestion des connaissances reposait sur l'idée de l'importance de suivre et documenter tous les processus de l'organisation afin que cette information soit accessible à tous, dans un environnement corporatif, par exemple, un Intranet. Les entreprises ont alors investi des sommes importantes dans différentes technologies comme des collecticiels qui devaient, justement, les aider à gérer et coordonner leurs activités. Cependant, cet effort s'est révélé peu productif, car on a réalisé que tout le travail nécessaire

à rendre accessible cette information était trop énorme par rapport au nombre de gens qui utilisaient réellement l'Intranet mis à leur disposition.

La seconde génération de la gestion de la connaissance s'est inspiré de la théorie des systèmes complexes et de celle du chaos. Le modèle classique d'organisation fermée « intégrée » a été remplacé par un modèle organisationnel compris en tant que système complexe à l'intérieur duquel des entités interdépendantes sont capables de réagir à leur environnement. Par exemple, des études (Baker et Gollub, 1990) ont montré qu'un système complexe fonctionne à son maximum lorsqu'il est *au bord du chaos*. Le *bord du chaos* (*edge of chaos)*, est un terme qui a été introduit par Chris Langton de l'institut Santa Fe et désigne l'état critique dans lequel se trouve le système au moment où une toute petite fluctuation pourrait le pousser soit vers un comportement chaotique, soit vers un comportement stable. Lorsqu'un système complexe est *au bord du chaos*, il est dans un état où les changements peuvent survenir facilement et de façon spontanée. Ainsi, l'identification et la prolongation de cet état ont des implications intéressantes au niveau de l'organisation des entreprises qui doivent constamment s'adapter et réagir rapidement au marché sans cesse changeant.

Cette nouvelle façon d'aborder la gestion de la connaissance donne la priorité à la manière dont les gens construisent et utilisent la connaissance. Elle s'intéresse aux interactions entre les personnes formant le réseau social qu'est l'organisation. Aujourd'hui, donc, les gestionnaires comprennent bien l'importance des connaissances dites tacites (connaissances personnelles, subjectives, intuitives, relevant de l'expérience, associées à un contexte spécifique) tant que celles, plus tôt convoitées par la première génération de gestion de la connaissance, dites explicites (codifiées, sémantiques, formelles, facilement communicables, extériorisées). Les connaissances tacites se communiquent par la socialisation. (Nonaka, 1991). Celles-ci se transmettent donc à travers les interactions humaines et plusieurs technologies sont maintenant employées à favoriser ces interactions. Ce sont les logiciels sociaux tels que le courrier électronique, Usenet, IRC, la messagerie instantanée, les blogues, les wikis, NNTP, folksonomy et les communautés virtuelles en ligne. On parle maintenant de l'âge de la connexion plutôt que de celui de l'information.

2.2.2 Les réseaux sociaux comme aide à la recherche d'information

Pour débusquer l'information pertinente, les réseaux sociaux s'avèrent souvent très utiles. La recherche d'information est généralement entravée par deux choses : la difficulté de faire une requête précise lorsque l'utilisateur ne connaît pas bien le sujet et le fait que beaucoup de connaissances sont tacites et donc difficilement accessibles. En général, la meilleure façon de remédier à ce problème est de demander une assistance humaine. Ainsi, de plus en plus de systèmes informatiques sont développés dans cette optique et utilisés pour créer des communautés virtuelles et faciliter la localisation de la bonne personne dans la recherche d'information (Memmi et Nérot, 2003).

Les méthodes classiques de recherche d'information ne tiennent pas compte de l'aspect social de la production et la consommation de l'information. De plus en plus, des techniques plus actuelles tentent d'intégrer des informations à propos de l'environnement social de l'utilisateur et de sa position dans le réseau social afin d'améliorer l'efficacité de sa recherche.

Par exemple, le moteur de recherche *Google* a été le premier à intégrer une analyse de la structure du graphe du Web dans ses algorithmes de recherche (Brin et Page, 1998). Comme on sait, Google renvoie en priorité les pages sur lesquelles pointent le plus de liens, en considérant que c'est un bon indice de leur crédibilité et de leur pertinence.

Le logiciel *Human Links* (http://www.human-links.com) est un moteur de recherche distribué, construit selon une architecture poste-à-poste (peer-to-peer), qui exploite l'expertise collective des utilisateurs. Lorsqu'un utilisateur se connecte au système, il devient un nœud dans ce réseau poste-à-poste (peer-to-peer) et de ce fait, contribue implicitement à la recherche d'information. Le système extrait automatiquement un profil spécifique pour chaque utilisateur à partir de ses marque-pages Web, par exemple (d'autres informations peuvent aussi être utilisées) et la similarité entre les profils des utilisateurs définit un réseau virtuel. Dans ce réseau, les utilisateurs sont considérés comme étant proches les uns des autres si leurs profils sont similaires et non parce qu'ils sont adjacents dans la structure réelle du graphe. Lorsqu'un utilisateur soumet une requête au système, cette requête est propagée à

travers le réseau poste-à-poste pour localiser des experts potentiels sur le sujet en comparant la requête avec les profils des utilisateurs. Lorsqu'un expert est trouvé, le système peut fonctionner de deux façons différentes : il peut demander à l'expert s'il est prêt à aider l'auteur de la requête, ou bien il peut utiliser le profil de l'expert pour raffiner la requête initiale puis reprendre sa recherche. Ce logiciel peut donc être utilisé pour trouver des documents aussi bien que de l'expertise humaine.

ReferralWeb (Kautz, Selman et Shah, 1997) est un autre exemple. Ce système a pour but d'améliorer la recherche d'information en construisant un réseau social puis en trouvant un expert dans ce réseau. Les liens sociaux sont extraits automatiquement du Web en observant les cooccurrences des noms propres cités dans les pages Web. Par exemple, les co-auteurs de publications dans les bibliographies ou les membres d'un même département universitaire seront considérés comme étant en relation les uns avec les autres. Ce réseau peut alors être utilisé pour localiser des experts sur un sujet particulier en suivant les liens sociaux.

Une discussion plus complète sur l'utilisation des réseaux sociaux comme aide à la recherche d'information est présentée dans l'article de (Kirsch, Gnasa et Cremers, 2006).

2.2.3 Les entreprises en réseaux

Pour soutenir la concurrence et maintenir un bon niveau de productivité, dans une société technologique où les produits sont v désuets, où les technologies évoluent constamment et où les entreprises doivent répondre rapidement à la demande, de plus en plus, « pour créer davantage de valeur et soutenir des gains de productivités, les entreprises devront s'ouvrir vers l'extérieur et identifier de nouvelles façons de coordonner leurs activités autour de réseaux réunissant un grand nombre d'entreprises. » (Poulin, 2004). La mise en commun des ressources a pour but de favoriser l'innovation et ainsi, permettre aux entreprises de demeurer concurrentielles. Dans cette optique, il serait avantageux, par exemple, d'étudier les types de structures relationnelles qui favorisent l'émergence de l'innovation.

2.2.4 Conception d'une future génération de l'Internet inspirée des réseaux sociaux

Il existe, en ce moment, une proposition qui pourrait bien devenir la future génération d'Internet : la création d'un réseau social augmenté (Augmented social network ASN) qui bâtirait identité et confiance à l'intérieur de l'architecture d'Internet. ASN (Jordan, Hauser et Foster, 2003) propose une forme de citoyenneté en ligne et a pour but de faciliter l'introduction entre personnes qui partagent les mêmes affinités ou des intérêts complémentaires à travers le réseau social. Il n'est pas certain que ASN devienne effectivement l'Internet du futur, mais cette proposition montre encore l'intérêt des réseaux sociaux dans l'actualité.

2.3 Analyse des structures relationnelles

2.3.1 Définition et représentation d'un réseau social

Un réseau social peut être défini comme étant un ensemble de relations entre un ensemble fini d'individus, ceux-ci faisant figure d'acteurs (ou d'agents) à l'intérieur du réseau.

Il est possible d'identifier plusieurs types de relations (collaboration, soutien, contrôle, conseil, influence, parenté, amitié, etc.) qu'entretiennent les acteurs les uns avec les autres au sein d'un ensemble social. À titre d'exemple, prenons les employés d'une entreprise. On peut alors se demander qui collabore avec qui au sein de cette société. On obtient alors un réseau social formé de l'ensemble des relations de collaboration entre les employés de cette entreprise. Dans cet exemple, le réseau est constitué de relations entre individus, mais nous pourrions, de la même façon, considérer les relations qui existent entre différents groupes sociaux. Dans ce cas, l'acteur n'est plus un individu, mais un ensemble d'individus. Par exemple, nous pourrions étudier les relations de collaboration entre plusieurs entreprises, chaque entreprise faisant figure d'acteur à l'intérieur du réseau.

Les réseaux sociaux, de par leur nature, peuvent facilement être représentés sous forme de graphe : chaque acteur du réseau social est représenté par un sommet (ou noeud) du graphe et

chaque relation entre deux individus est représentée par une arête (si la relation est bidirectionnelle) ou un arc (si la relation est unidirectionnelle) du graphe. Par analogie, il est donc possible d'appliquer les concepts de la théorie des graphes aux réseaux sociaux. On peut alors parler de proximité, de distance, de densité, etc.

Par exemple, il existe plusieurs mesures de distance. L'une d'entre elles est la distance géodésique qui mesure le plus court chemin entre deux sommets. Lorsque la distance géodésique entre deux acteurs est faible, on dit que les acteurs sont proches l'un de l'autre, si elle est grande, on dit qu'ils sont éloignés l'un de l'autre. Le diamètre d'un graphe correspond à la longueur du plus long géodésique reliant n'importe quelle paire de sommets dans le graphe. C'est la distance géodésique maximale entre deux membres.

Aussi, on dit d'une relation qu'elle est directe si la distance entre les deux acteurs est d'un seul pas et qu'elle est indirecte sinon. Sachant que dans un graphe de n sommets, le nombre de liens possibles est égal à $n(n-1)$ pour un graphe orienté et à $n(n-1)/2$ pour un graphe non orienté, on peut alors calculer la densité d'un graphe de cette façon :

Densité = nombre de relations directes / nombre de relations possibles

Ceci n'est qu'un bref aperçu des applications possibles de la théorie des graphes aux réseaux sociaux, mais ce qu'il faut retenir ici est que l'étude des ensembles sociaux sous forme de réseaux permet l'application de mesures qui peuvent s'avérer très utiles lors de l'analyse de la structure relationnelle. De plus, l'information contenue dans un graphe peut être représentée dans une matrice. La représentation matricielle permet alors l'utilisation d'opérations algébriques matricielles dans l'analyse des réseaux. On appelle matrice sociométrique une matrice qui montre tous les arcs (relations directes) d'un graphe. Pour connaître la distribution des relations indirectes de x pas, il suffit d'élever cette matrice à la puissance x. La figure 2.3.1.2 illustre ce cas.

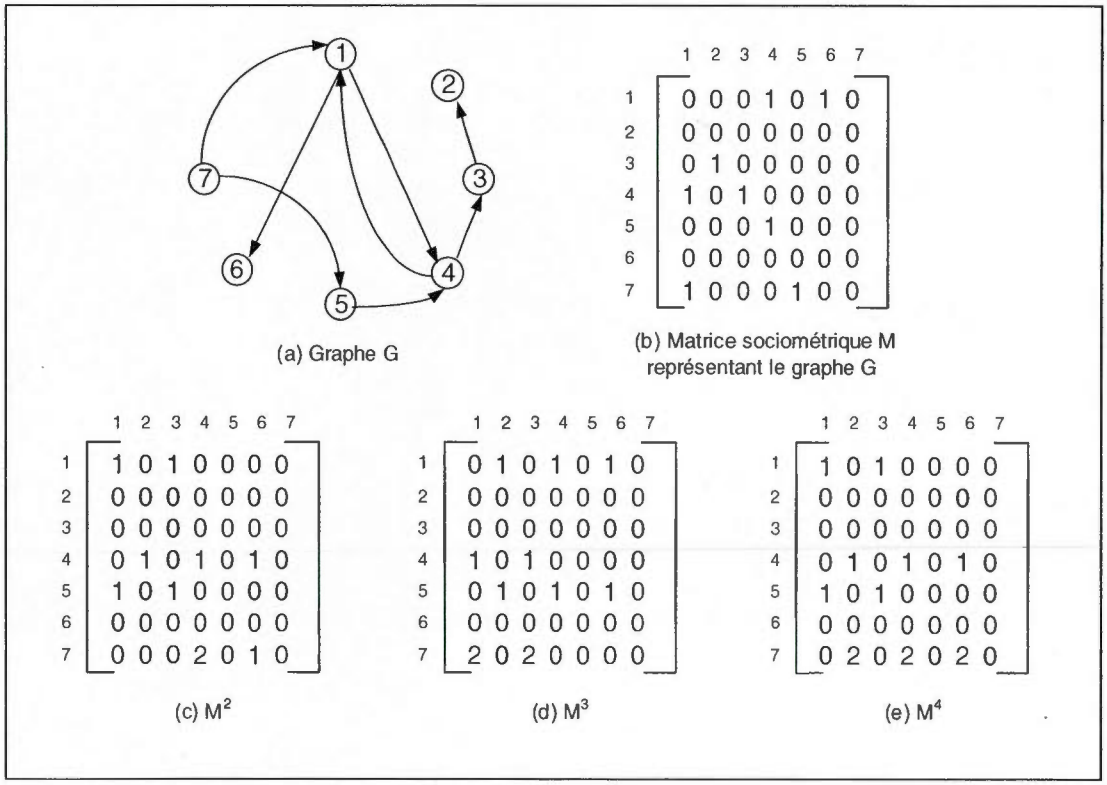


Figure 2.3.1.2 Chemins de longueur x dans une matrice sociométrique
La matrice sociométrique de la figure (b) montre tous les chemins de longueur 1 contenus dans le graphe G de la figure (a). Il suffit d'élever cette matrice à la puissance x pour obtenir une matrice du nombre de chemins de longueur x pour aller du $i^{ième}$ sommet (ligne) vers le $j^{ième}$ sommet (colonne), pour tous les sommets du graphe. Par exemple, la figure (c) montre la matrice M élevée au carré. Cette matrice contient donc tous les chemins de longueur 2. Ainsi, on peut y observer qu'il existe 2 chemins de longueur 2 pour aller du sommet 7 au sommet 4 car $M^2[7][4] = 2$. La même chose peut être observée dans les figures (d) et (e) pour les matrices M^3 et M^4 qui montrent respectivement le nombre de chemins de longueur 3 et le nombre de chemins de longueur 4 allant d'un sommet à un autre pour tous les sommets du graphe.

2.3.2 L'approche structurale

Dans le champ de la sociologie, l'approche structurale (Lazega, 1998) est une méthode qui a été développée pour étudier les réseaux sociaux afin de mieux comprendre les phénomènes sociologiques. Celle-ci tente de formaliser et de systématiser les relations entre les acteurs à l'aide d'outils mathématiques ou statistiques, ce qui la distingue d'autres méthodes utilisées en sociologie qui sont souvent plus informelles et plus qualitatives.

L'approche structurale propose donc diverses mesures pour l'analyse des réseaux relationnels et nous en verrons quelques-unes dans les sections suivantes.

2.3.2.1 Procédures de repérages de sous-groupes cohésifs

Ces procédures permettent de représenter et de décrire le système au niveau macro en fournissant des techniques de partitionnement du réseau en sous-ensembles. La description des relations entre les sous-ensembles donne une flexibilité qui permet un va-et-vient constant entre le niveau structural et le niveau local. Un sous-groupe cohésif est un sous-ensemble d'acteurs fortement reliés entre eux et la sociologie structurale propose plusieurs méthodes pour repérer différents types de sous-groupes dans la structure globale. Quelques-unes de ces méthodes sont expliquées ci-dessous.

Méthode sans l'usage de critères fixes

Cette méthode repose sur la comparaison entre la fréquence relative des relations entre membres d'un sous-groupe (relations internes) et celles entre membres et non-membres (relations externes). C'est la proportion donnée par la division de la densité des relations internes par la densité des relations internes et externes. Ces sous-groupes existent parce qu'ils sont relativement plus cohésifs en comparaison au reste du réseau.

Ces sous-groupes cohésifs sont construits par l'addition successive de membres à un sous-ensemble initial tant que la proportion mesurée ne varie pas trop.

Méthode basée sur la réciprocité complète

Cette méthode repose sur la définition de cliques. Les cliques sont des sous-ensembles de membres tous adjacents les uns aux autres (sous-graphe maximum complet de trois sommets ou plus). Cela signifie que les membres d'une même clique se choisissent tous entre eux. Si un membre appartient à plusieurs cliques, on parle de cliques superposées.

Cette façon de définir un sous-groupe cohésif est très sévère, car l'absence d'un seul arc peut empêcher un graphe d'être une clique. Ce phénomène se produisant assez souvent dans les réseaux dispersés, par exemple, cette méthode risque d'être inutilisable dans plusieurs cas. Les méthodes suivantes sont moins strictes.

Méthode basée sur l'accessibilité et le diamètre

Cette méthode repose sur la définition de la n-clique. Celle-ci est définie comme un sous-graphe maximal dans lequel la distance géodésique la plus grande qui sépare deux sommets n'est pas supérieure à n. L'important, ici, contrairement aux cliques, n'est donc pas que les membres soient tous adjacents, mais qu'ils soient accessibles les uns aux autres, sur une courte distance.

Cette méthode est donc moins sévère que la méthode basée sur la réciprocité complète, mais présente aussi ses limites. En effet, le chemin qui mène un membre de la clique à un autre membre peut passer par un ou même plusieurs membres à l'extérieur de la clique et le sous-groupe ainsi formé peut ne pas être aussi cohésif qu'on le voudrait. Par contre, il existe certaines solutions à ce problème :

n-clan : consiste à exclure toutes les cliques qui contiennent des géodésiques dont l'un des membres n'est pas un membre du sous-groupe.

n-club : consiste à définir un sous-graphe maximal de diamètre n dans lequel on n'inclut aucun membre qui soit à une distance géodésique inférieure ou égale à n de tous les membres du sous-graphe.

Méthode basée sur le nombre de membres adjacents

Cette approche se fonde sur la définition d'un nombre minimum d'acteurs adjacents à tous les autres membres du sous-ensemble. Il n'est donc pas nécessaire que tous soient reliés à

tous, mais plutôt que tous les membres du sous-groupe aient des liens adjacents avec « beaucoup » d'autres membres.

Cette méthode est plus robuste et moins vulnérable que la méthode des n-cliques, car si l'on retire un membre du sous-groupe, la structure n'est pas modifiée contrairement à une n-clique qui elle, n'existerait plus.

Pour construire ce genre de sous-groupes, il existe deux approches :

k-plex : spécifie le nombre de liens qui peuvent manquer.

k-cores : spécifie le nombre de liens obligatoires.

Méthode de l'équivalence structurale

La méthode de l'équivalence structurale se distingue des techniques d'identification de sous-groupes cohésifs mentionnées précédemment. En effet, celle-ci cherche davantage à définir des sous-ensembles d'acteurs ayant le même profil relationnel qu'à identifier des sous-groupes d'acteurs qui ne font qu'interagir entre eux.

L'équivalence structurale cherche donc à déterminer les sous-groupes d'acteurs qui jouent des rôles similaires dans le réseau, qui ont des relations comparables avec d'autres acteurs. Dans cet ordre d'idées, la substitution de deux acteurs structuralement équivalents, dans le réseau, ne devrait pas faire de différence.

L'équivalence structurale, telle qu'elle vient d'être définie, est une propriété qu'il est pratiquement impossible d'observer dans la vie sociale. Deux personnes n'ont jamais le même profil relationnel : ils n'ont jamais exactement, par exemple, les mêmes amis ou ennemis. Pour cette raison, l'équivalence structurale se calcule de façon statistique. Il existe plusieurs techniques de mesure de l'équivalence structurale, dont les modèles catégoriques (block models) et l'utilisation de la distance euclidienne (Lazega, 1998).

Méthode de l'équivalence régulière

Le but de l'analyse des sous-groupes fondée sur l'équivalence structurale est de repérer des sous-ensembles d'acteurs au profil relationnel semblable c'est-à-dire, qui ont les mêmes liens avec les mêmes membres du reste du système. Il s'ensuit donc qu'une telle analyse ne permet pas de déceler des statuts et des rôles qui ne soient pas exclusivement spécifiques à la population étudiée. Cette méthode ne peut donc pas être utilisée pour comparer deux populations différentes. C'est pour tenter de dépasser cette limite que l'équivalence régulière a été introduite.

L'équivalence régulière est, en quelque sorte, une forme plus relâchée d'équivalence. Contrairement à l'équivalence structurale, l'équivalence régulière n'exige pas qu'un individu ait les mêmes relations avec tous les membres du système. Deux acteurs sont dits régulièrement équivalents s'ils ont au moins une relation avec des individus eux-mêmes équivalents. On produit ainsi des classes corrélatives qui se définissent les unes par rapport aux autres. Par exemple, la classe « enseignant » est définie par rapport à la classe « étudiant » (et vice versa) et, selon l'équivalence régulière, il suffit de n'avoir qu'un seul étudiant pour être considéré comme faisant partie de la classe des enseignants.

Cependant, cette méthode a aussi ses limites. La pratique de cette forme d'analyse peut, en effet, engendrer des classes si abstraites qu'on peut difficilement les retrouver dans la réalité des systèmes sociaux étudiés.

2.3.2.2 Procédures de positionnement des acteurs dans la structure

Les mesures de centralité et de prestige sont des mesures de la position relative des acteurs au sein d'un système social. Elles servent à identifier les acteurs les plus importants qui sont les plus susceptibles de contrôler l'allocation des ressources.

Centralité

Un acteur est très central lorsqu'il est engagé dans beaucoup de relations (directes ou indirectes). Ici, la direction des arcs ne compte pas, donc cette mesure peut être utilisée pour des relations orientées ou non.

Soit g, le nombre de sommets d'un graphe (nombre d'acteurs dans le réseau).

Centralité de degré C_d (*degree*) : Taille du réseau de l'acteur (nombre de liens)

Interprétation : Plus un acteur est central, plus il est actif dans le système.

$C_{di} = \sum_j x_{ij}$ où x_{ij} est la valeur du lien de i à j

$C'_{di} = \frac{\sum_j x_{ij}}{g-1}$ (normalisation)

Centralité de proximité C_c (*closeness*) : Nombre minimum de pas que l'acteur doit effectuer pour entrer en contact avec les autres acteurs du système. On ne peut pas calculer cet indice lorsque le réseau comprend plusieurs composantes connexes.

Interprétation : mesure d'autonomie, d'indépendance à l'égard du contrôle exercé par d'autres.

$Cc_i = \frac{1}{\sum_{j=1}^{g} d_{ij}}$, pour tous les $j \neq i$

où d_{ij} est la distance géodésique entre les acteurs i et j et où $\sum_{j=1}^{g} d_{ij}$ est la distance totale entre l'acteur i et tous les autres acteurs du système. Au maximum, cet indice = $\frac{1}{(g-1)}$ lorsque l'acteur est adjacent à tous les autres acteurs.

$$C'c_i = \frac{g-1}{\sum_{j=1}^{g} d_{ij}} = (g-1)Cc_i \quad \text{(normalisation)}$$

$C'c_i$ vaut 1 lorsque l'acteur i est adjacent à tous les autres.

Centralité d'intermédiarité C_B (*betweeness*) : Proportion des géodésiques entre j et k qui passent par i.

Interprétation : Contrôle exercé par l'acteur sur les interactions entre deux autres acteurs. Lorsque deux acteurs ne sont pas adjacents, ils dépendent d'autres acteurs pour leurs échanges et ces acteurs intermédiaires peuvent avoir la capacité de bloquer la circulation des ressources. Plus un acteur se trouve au milieu et constitue un point de passage nécessaire sur des chemins que d'autres doivent utiliser pour communiquer entre eux, plus il est central de ce point de vue.

$$C_{Bi} = \frac{\sum_{j<k} g_{jk}(i)}{g_{jk}}$$

pour $i \neq j,k$ où g_{jk} = l'ensemble des géodésiques entre j et k et où $g_{jk}(i)$ = un chemin entre j et k passant par i.

Le minimum = 0 si i ne tombe sur aucun géodésique.

Le maximum = $\frac{(g-1)(g-2)}{2}$ si l'acteur se trouve sur toutes les géodésiques.

$$C'_{Bi} = \frac{C_{Bi}}{\frac{(g-1)(g-2)}{2}} \text{ (normalisation)}$$

Les différentes mesures de centralités déterminent l'importance d'un acteur dans le réseau social. La figure 2.3.2.2.1 illustre le « Kite Network » développé par David Krackhardt (tiré de « How to do Social Network », www.orgnet.com/sna.html). Cette figure illustre très bien les différentes mesures de centralité. On observe que dans ce réseau, Lucie obtient le meilleur résultat pour la centralité de degré, car c'est elle qui possède le plus de liens directs. Cependant, le fait d'avoir plusieurs relations n'est pas toujours aussi important que les personnes avec qui on entretient ces relations. En effet, on remarque que les contacts de Lucie font tous partie de sa propre clique, ils sont tous reliés entre eux. Il aurait été plus avantageux pour Lucie d'avoir des contacts hors clique afin de devenir un intermédiaire obligé entre les membres de sa clique et ces contacts extérieurs.

De son côté, Manon possède peu de liens directs, moins que la moyenne dans le réseau. Son score de centralité de degré est donc relativement petit par rapport aux autres membres du réseau. Par contre, du point de vue de la centralité d'intermédiarité, elle occupe une des meilleures positions dans le réseau. Elle joue le rôle de l'unique intermédiaire entre les membres de la clique de gauche et les deux membres à sa droite. Elle peut ainsi contrôler les communications entre ces deux parties. Elle est en position de pouvoir, car sans elle, Justin et Louise ne pourraient pas avoir accès à l'information et aux connaissances provenant de la clique de Lucie. Un acteur ayant un fort degré de centralité d'intermédiarité a beaucoup d'influence sur la circulation des connaissances dans le réseau.

Jean et Jim possèdent un peu moins de connexions que Lucie cependant, la structure de leurs connexions directes et indirectes leur permet d'avoir accès à tous les autres acteurs du réseau plus rapidement que n'importe quel autre membre. Ils ont accès aux plus courts chemins vers tous les autres acteurs. Ils sont proches de tout le monde et possèdent donc le score de

centralité de proximité le plus élevé. Ils sont alors dans une excellente position pour exercer un contrôle sur le flux d'information dans le réseau, car ils ont la meilleure visibilité de ce qui y circule.

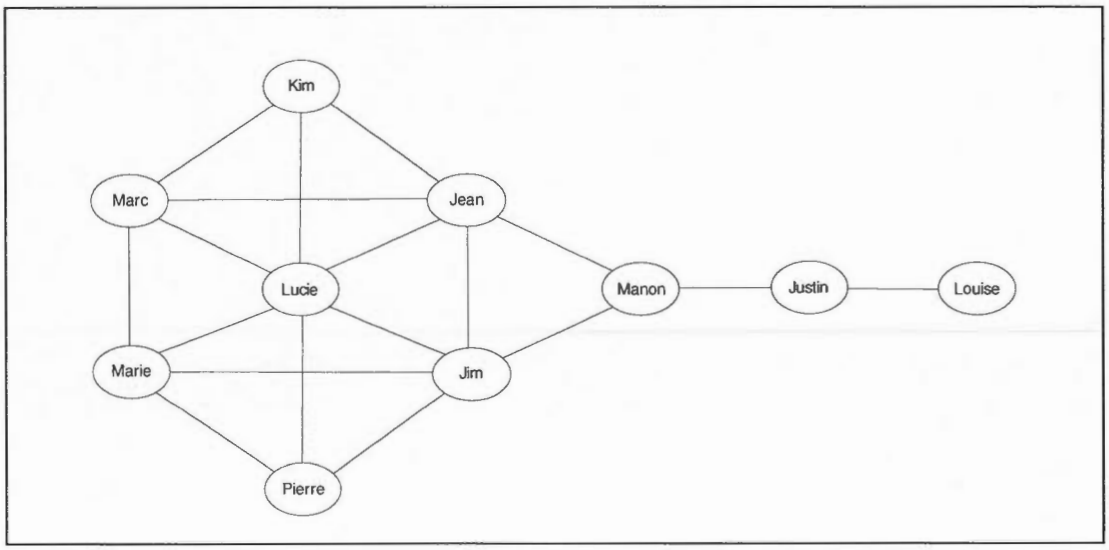


Figure 2.3.2.2.1 Illustration du « Kite Network » développé par David Krackhardt

Prestige

Prestige de degré : Un acteur est prestigieux lorsqu'il reçoit beaucoup de choix. Ici, la direction des arcs a son importance donc cet indice ne peut être calculé que pour les relations orientées. Le prestige de degré correspond au calcul du demi-degré intérieur. C'est le nombre de choix reçus (arcs entrants).

Interprétation : Plus cet indice est grand, plus l'acteur est populaire.

Prestige par proximité : Est une extension du prestige de degré. Ce dernier ne compte que les acteurs adjacents à i mais on peut généraliser cet indice en définissant un « domaine d'influence de i » et utiliser les relations directes et indirectes à l'intérieur de ce domaine pour calculer le prestige par proximité.

Interprétation : Plus cet indice est grand, plus l'acteur est populaire.

Prestige basé sur le statut ou le rang : Si un acteur est en relation avec des membres eux-mêmes prestigieux, cela augmente son propre prestige.

2.3.2.3 Une procédure d'association entre positions et comportements des acteurs

Les structures relationnelles entre les acteurs et la position de ceux-ci dans le réseau influencent leur comportement. La description des structures relationnelles cherche à mettre en lumière les contraintes que la structure fait peser sur le comportement des acteurs, la manière dont les acteurs gèrent ces contraintes et les processus de restructuration qui peuvent en résulter. De ce point de vue, on peut s'interroger sur la composition particulière des structures relationnelles qui contraignent davantage les acteurs ou qui, au contraire, leur offrent davantage d'opportunités. La structure relationnelle peut donc procurer aux acteurs du pouvoir, comme on l'a vu avec la centralité et le prestige, mais aussi, certaines formes d'autonomie.

Autonomie

L'autonomie est la capacité de pouvoir substituer une relation à une autre, d'avoir une alternative relationnelle. On peut attribuer un score d'autonomie à chaque acteur du réseau en comptant, par exemple, le nombre de contacts non redondants de chacune de ses relations. Un acteur qui possède beaucoup de contacts non redondants n'est pas en position d'autonomie. Cette mesure cherche à voir qui, dans le réseau, a le pouvoir de ne pas se laisser confiner dans une relation incontournable. La figure 2.3.2.3.1 (a) montre le sous-réseau de l'acteur *i*. On voit qu'il n'y a aucune relation entre les différents contacts de *i* ce qui procure à cet acteur un grand pouvoir d'action étant donné qu'il est le seul point de communication possible pour que les acteurs *c1* à *c8* puissent communiquer entre eux. L'acteur *i* devient une relation incontournable et peut alors contrôler plus facilement les ressources qui circulent en passant nécessairement par lui. Il a une grande autonomie. Les acteurs *c1* à *c8*, par ailleurs, ne sont pas très autonomes, car ils ne possèdent aucune alternative relationnelle pour se rejoindre les uns les autres. La figure 2.3.2.3.1 (b) montre une structure relationnelle similaire, mais dans laquelle l'acteur *j* possède les relations

illustrées par les lignes de tirets dans la figure. Dans cet exemple, pour communiquer avec les contacts *c2*, *c3*, *c4*, *c6* et *c7*, l'acteur *j* n'est pas dans l'obligation de passer par *i*. Il a donc une alternative relationnelle pour rejoindre ces contacts et possède, de ce fait, une certaine autonomie. Par contre, *j* doit absolument passer par *i* s'il veut communiquer avec *c1* et *c5*. De son côté, l'acteur *i* a beaucoup moins de pouvoir sur la circulation des ressources que dans la structure relationnelle montrée en (a), car étant donné que plusieurs de ses contacts directs peuvent communiquer entre eux, ils peuvent aussi s'organiser et offrir une certaine résistance.

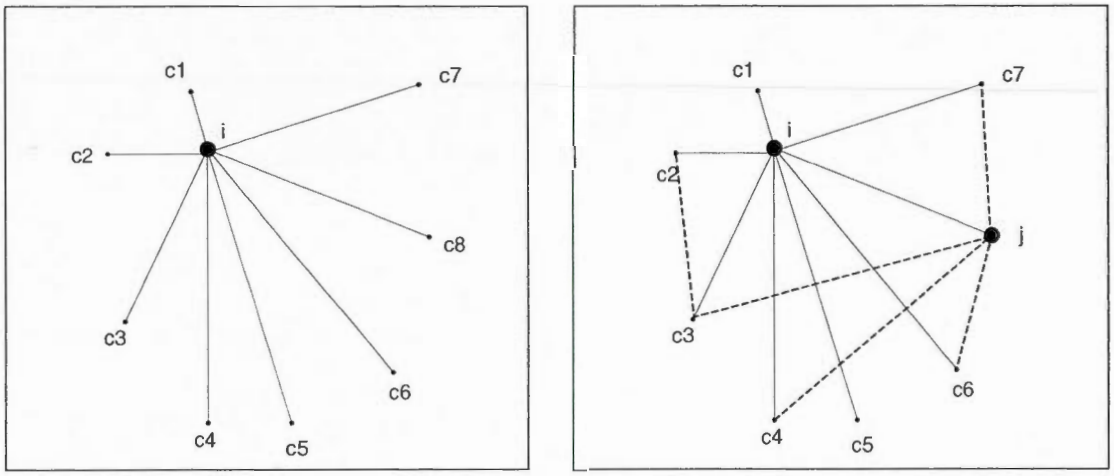


Figure 2.3.2.3.1 Autonomie des acteurs dans différentes structures relationnelles.

Plus un acteur possède de contacts non redondants dans le réseau, plus ses relations sont uniques, plus il risque d'être paralysé par la structure relationnelle. D'autre part, plus il est relié à des contacts non reliés entre eux, plus il y a de « trous » dans la structure, plus il est autonome, car les individus qui l'entourent ne sont pas organisés entre eux. Ainsi, la structure des relations d'un acteur peut augmenter ou limiter sa liberté d'action et lui procurer plus ou moins d'autonomie.

2.4 Revue des modèles actuels

Les modèles actuels de réseaux sociaux proviennent de plusieurs disciplines telles que la physique, les mathématiques, la sociologie, l'économie et la psychologie sociale. Dans cette

section, sans faire une énumération exhaustive de tous les modèles existants, nous présentons quelques modèles importants provenant de domaines divers.

2.4.1 Modèles théoriques

Une grande partie des modèles sociaux viennent de la physique avec l'adaptation de lois physiques à la modélisation des systèmes sociaux. Ces modèles se concentrent plus particulièrement sur les propriétés de la structure globale des réseaux sans tenir compte des différences de types de liens ou des propriétés des agents. Comme exemple de modèles théoriques, nous présentons ici les automates cellulaires, les petits mondes, le modèle de réseau indépendant de l'échelle « scale-free » et finalement, un modèle de réseaux d'amis.

Automate cellulaire

Le champ de la modélisation sociale a hérité de plusieurs outils provenant des mathématiques, de la physique et en particulier, des automates cellulaires (Wolfram, 1986). La structure d'interactions sous-jacente à ces modèles est, en général, une grille régulière dans laquelle chaque case représente une cellule et où l'environnement social est représenté par la grille entière. Les voisins de chaque cellule sont alors déterminés par leur proximité spatiale sur la grille, selon un voisinage de Von Neumann ou de Moore, par exemple. La figure 2.4.1.1 illustre ces deux types de voisinage.

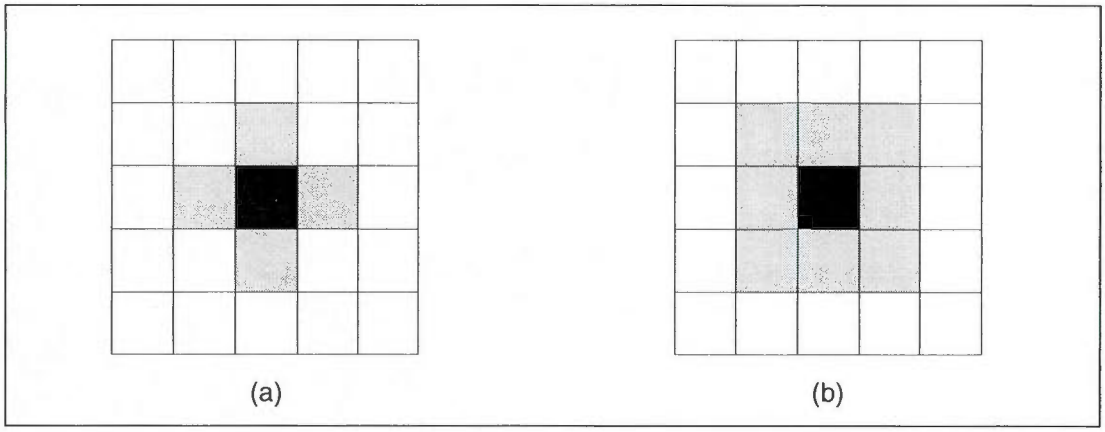


Figure 2.4.1.1 Voisinage de Von Neumann et voisinage de Moore dans un automate cellulaire
Dans cette figure, les voisins de la cellule noire sont représentés en gris. (a) Le voisinage de Von Neumann relie une cellule à ses quatre voisins (nord, sud, est, ouest). (b) Le voisinage de Moore ajoute quatre voisins de plus que celui de Von Neumann (nord-est, sud-est, sud-ouest et nord-ouest).

Le jeu de la vie (The Game Of Life) est un exemple populaire d'automate cellulaire qui a été inventé par le mathématicien John Conway. Ce jeu consiste en une collection de cellules qui, se basant sur quelques règles mathématiques simples, peuvent vivre, mourir ou se reproduire. Étant donné les conditions initiales, les cellules forment divers motifs à mesure que le jeu se poursuit. Les règles se présentent comme suit :

Pour un espace occupé sur la grille :

1. Chaque cellule avec moins de deux voisins meurt de solitude.
2. Chaque cellule avec plus de trois voisins meurt d'une surpopulation.
3. Chaque cellule avec deux ou trois voisins survit.

Pour un espace non occupé sur la grille :

4. Chaque cellule vide entourée de 3 voisins naît (la case devient occupée).

Pour généraliser ce type de modèle, (Flache et Hegselmann, 2001) proposent différentes structures de grilles régulières formées, par exemple, de cellules triangulaires ou hexagonales et d'autres irrégulières déterminées d'après un diagramme de Voronoï.

Le problème principal rencontré dans cette approche est qu'on part d'une mosaïque bidimensionnelle et qu'on dérive ensuite, de cette mosaïque, la localisation des cellules et de leur voisinage. Cette méthode peut s'avérer limitative pour modéliser des systèmes sociaux qui ne sont pas seulement situés dans un espace géographique. De plus, dans ce genre de modèle, on ne peut pas modifier facilement la connectivité moyenne du graphe, car il y a un nombre fixe de quatre ou huit voisins pour les voisinages de Von Neumann ou de Moore. En ce qui concerne les grilles générées par un diagramme de Voronoï, il est possible d'ajuster la connectivité moyenne du graphe, mais à l'intérieur de certaines limites.

Les petits mondes

L'effet des petits mondes (small world effect) a été largement étudié. La première expérience conduite en ce sens fut celle de (Milgram, 1967) qui donna lieu à l'expression bien connue des six degrés de séparation (Guare, 1990). Cette expression signifie que le nombre de contacts interposés entre une personne choisie au hasard et n'importe quelle autre dans le monde est d'environ six. De nombreuses autres études ultérieures prouvèrent aussi que si le nombre six n'est pas nécessairement le nombre juste, il n'en demeure pas moins que la chaîne de contacts qui relie entre eux deux individus, choisis aléatoirement sur la planète, est très petite par rapport à la population entière : si l'on calcule la distance qui sépare chacun d'entre nous de n'importe quel autre individu sur la planète, en terme de nombre de contacts interposés pour se rendre de l'un à l'autre, le monde est petit (Korte et Milgram, 1970).

Certains scientifiques ont donc essayé de définir la structure d'interactions de leur modèle pour représenter cet effet des petits mondes observé dans la réalité.

Le modèle le plus simple est le graphe aléatoire. Les graphes aléatoires ont largement été étudiés en mathématique, en particulier par (Erdös et Rényi, 1960).

On suppose une population de N individus et le nombre z, appelé le nombre de coordination du réseau, est le nombre moyen de contacts que possède chaque individu du réseau. Il y a

donc $\frac{1}{2}Nz$ connexions symétriques au total dans le réseau. On peut construire un tel graphe en traçant N sommets et en reliant $\frac{1}{2}Nz$ paires de sommets choisis aléatoirement. La figure 2.4.1.2 illustre un graphe aléatoire de 6 sommets et ayant une connectivité moyenne de 2.

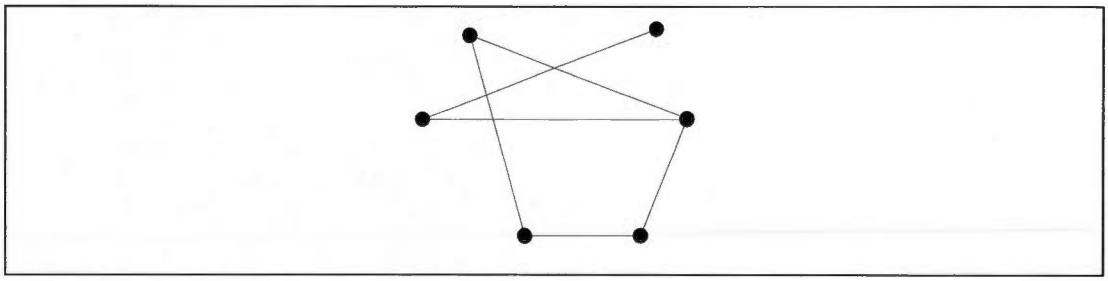


Figure 2.4.1.2 Graphe aléatoire avec $N = 6$ et $z = 2$

Il est facile de montrer que ce genre de réseau présente l'effet des petits mondes. En effet, supposons que A est un des sommets d'un graphe aléatoire. A possède donc z voisins qui possèdent à leur tour z voisins et ainsi de suite. Il s'ensuit que A possède z voisins directs, z^2 deuxièmes voisins, z^3 troisièmes voisins etc. Dans la vie courante, un individu a en moyenne entre 100 et 1000 connaissances ainsi, z^4 se trouve entre 10^8 et 10^{12}, ce qui est comparable à la population mondiale actuelle. Sachant que la distance entre deux nœuds est la longueur du plus court chemin qui relie ces nœuds, on définit le diamètre d'un graphe comme la plus grande distance entre deux de ses sommets. Le diamètre D d'un graphe aléatoire est donné par $z^D = N$ qui implique que $D = \log N / \log z$. On voit que D augmente très lentement avec l'augmentation de la population N.

Outre l'effet des petits mondes, une autre caractéristique importante observée dans les systèmes sociaux est la transitivité (clustering) qui signifie que deux individus qui connaissent une même personne ont de fortes chances de se rencontrer. En d'autres mots, dans un réseau social, si A connaît B et C, la probabilité que B et C se connaissent est beaucoup plus élevée que la probabilité que deux personnes simplement choisies au hasard se connaissent, car ils ont un contact commun qui est A.

Bien que les graphes aléatoires montrent parfaitement la propriété des petits mondes, ceux-ci ne représentent pas la propriété de transitivité que l'on peut observer dans la plupart des réseaux sociaux.

Supposons alors un graphe représenté par un treillis régulier à une dimension, pouvant être circulaire, dans lequel chaque sommet est relié à ses z voisins les plus proches. La figure 2.4.1.3 montre un tel graphe avec $z = 4$.

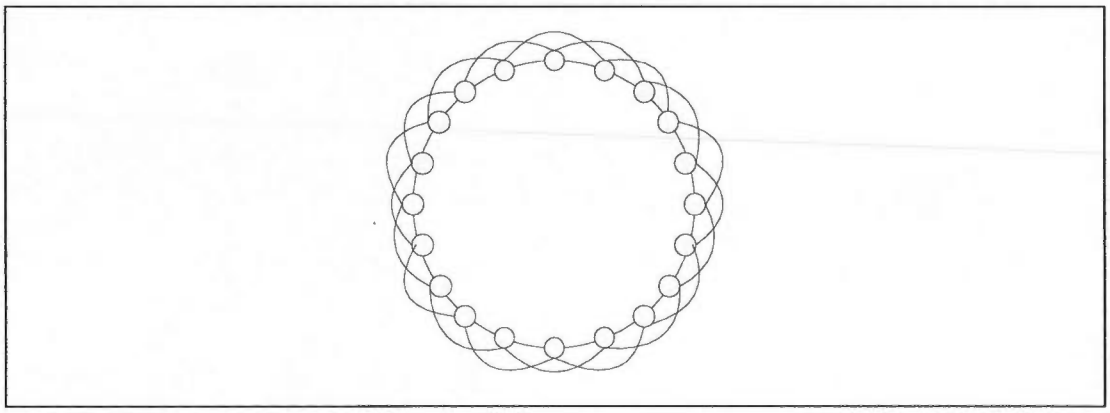


Figure 2.4.1.3 Treillis régulier avec $z = 4$

Dans la figure 2.4.1.3, on remarque que beaucoup des voisins immédiats de chaque sommet sont aussi des voisins l'un de l'autre. Ce type de graphe représente donc la propriété de transitivité.

Le coefficient de transitivité C est la fraction moyenne de paires de voisins d'un nœud qui sont aussi des voisins l'un de l'autre. Dans un graphe complet, où tout le monde connaît tout le monde, $C = 1$. Dans un graphe aléatoire, $C = z / N$, ce qui est très petit pour un grand réseau.

Pour ce qui est d'un graphe similaire à la figure 2.4.1.3, le coefficient de transitivité $C = \frac{3(z-2)}{4(z-1)}$ tend vers $\frac{3}{4}$ pour des z très grands (Newman, 1999).

D'un autre côté, ce type de graphe ne représente pas du tout l'effet des petits mondes, car la distance moyenne séparant deux sommets augmente de façon linéaire avec la taille du système.

Cependant, (Watts et Strogatz, 1998 ; Watts, 1999) ont proposé des algorithmes pour construire un graphe montrant à la fois l'effet des petits mondes et la propriété de transitivité. Le premier modèle, le modèle α, se construit comme le modèle de Erdös et Rényi, mais lors de l'ajout de liens au graphe, plutôt que de choisir deux nœuds de façon tout à fait aléatoire, on les choisit avec une probabilité qui augmente proportionnellement au nombre de contacts communs que possèdent les deux noeuds que l'on veut relier. Cela conduit rapidement à la formation de sous-groupes cohésifs (clusters) dans la structure. Le deuxième modèle, le modèle β, part d'une structure en forme de treillis régulier circulaire, mais y ajoute un certain degré de hasard. Pour ce faire, parmi tous les liens du graphe de la figure 2.4.1.3, on choisit quelques liens selon une probabilité p et, pour chaque lien choisi, on déplace une des extrémités de ce lien vers un autre sommet déterminé aléatoirement. La figure 2.4.1.4 montre un exemple de graphe obtenu par cet algorithme.

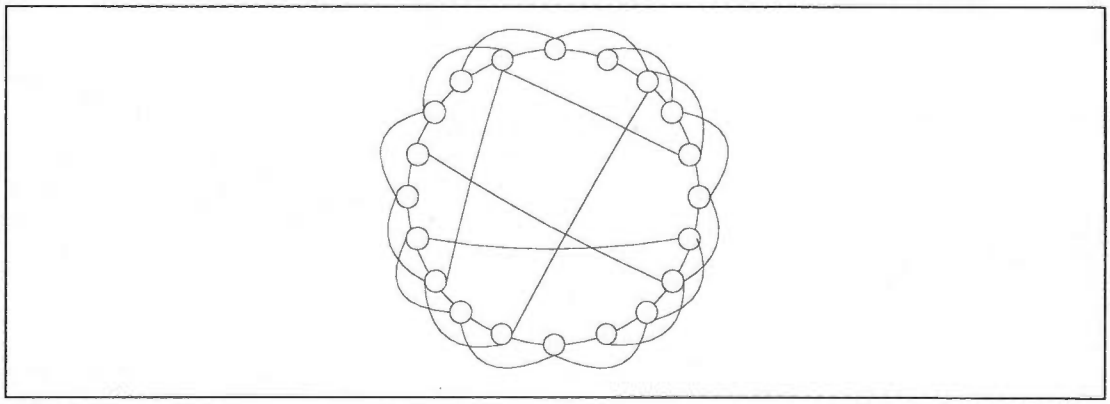


Figure 2.4.1.4 Modèle β exposant l'effet des petits mondes et la propriété de transitivité

La figure 2.4.1.4 montre que, pour une petite probabilité p, le treillis obtenu demeure quand même assez régulier, mais possède désormais quelques raccourcis qui le traversent et provoquent ainsi l'effet des petits mondes. En effet, Watts et Strogatz ont montré, par simulation numérique, que la distance moyenne entre deux sommets, lorsque la probabilité

$p = ¼$, est égale à 3.6. Ce résultat est à peine supérieur à la distance moyenne égale à 3.2, calculée pour un graphe aléatoire.

De plus, le nombre de voisins d'un sommet particulier peut être supérieur ou inférieur à z, mais en moyenne, le nombre de coordination est toujours égal à z. Clairement, la valeur du coefficient de transitivité C, pour de petites valeurs de p et un grand z, sera proche de celui calculé pour un treillis parfaitement régulier, comme celui de la figure 2.4.1.3. Ce modèle montre donc l'effet des petits mondes ainsi que la propriété de transitivité observés dans plusieurs systèmes sociaux réels.

Autres modèles de petits mondes

L'effet des petits mondes, selon la vision de Watts et Strogatz, se produit grâce à quelques raccourcis aléatoires, passant à travers le treillis. (Dorogovtsev et Mendes, 1999) ont trouvé une autre façon de provoquer ce phénomène. Il suffit d'ajouter au graphe quelques sommets qui ont un nombre de coordination inhabituellement plus élevé que les autres nœuds c'est-à-dire, qui sont reliés aléatoirement à un plus grand nombre de voisins que les autres nœuds du graphe. Ces quelques sommets particuliers deviennent alors des points de passage entre deux nœuds quelconques éloignés l'un de l'autre. Cet arrangement produit le phénomène des petits mondes tout en conservant la propriété de transitivité qu'offre la structure en treillis. La figure 2.4.1.5 montre à quoi ressemble un tel graphe.

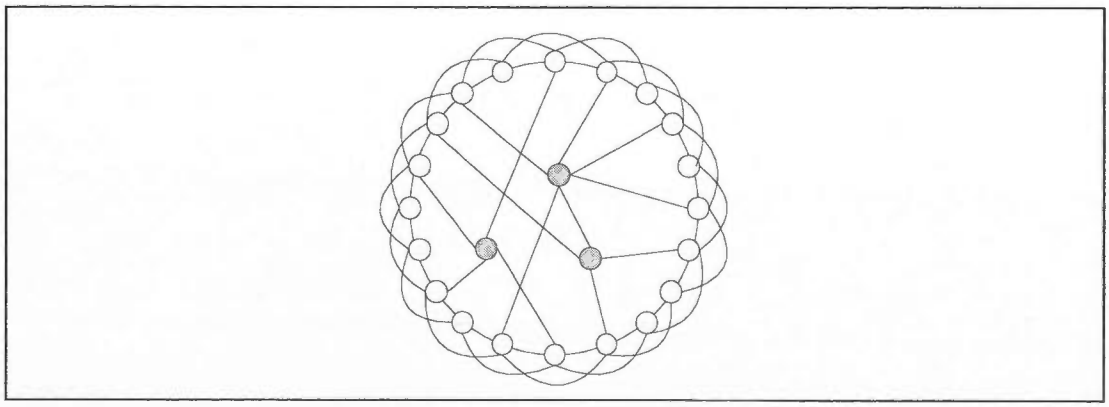


Figure 2.4.1.5 Modèle de petits mondes de Dorogovtsev et Mendes

Un autre modèle de petits mondes à été proposé par (Kleinberg, 1999) dont l'argument en défaveur du modèle de Watts et Strogatz porte sur le fait qu'un tel modèle, basé sur des connexions arbitraires entre des paires de sommets et selon une probabilité uniforme, est une piètre représentation des situations qu'on retrouve dans le monde réel. Il propose alors un modèle basé sur un treillis carré à deux dimensions dans lequel on a ajouté aléatoirement quelques connexions entre des paires de sommets i,j, mais selon une probabilité non uniforme égale à d_{ij}^{-r} suivant une loi de puissance où r est le coefficient caractéristique qui détermine l'allure de la courbe. Cette probabilité est fonction de la distance d_{ij} entre i et j où $d_{ij} = | x_i - x_j | + | y_i - y_j |$ et où (x_i, y_i) et (x_j, y_j) sont les coordonnées des sommets i et j dans le treillis.

Il semble possible, en effet, qu'un tel réseau représente mieux les interactions sociales que le modèle présenté par Watts et Strogatz même si les propriétés des deux modèles sont similaires en ce qui concerne l'effet des petits mondes et la transitivité.

Modèle indépendant de l'échelle (*scale-free*)

Une autre propriété de certains réseaux sociaux et plus particulièrement du WWW a été démontrée dans une étude conduite par (Albert, Jeong et Barabási, 1999). Ils ont remarqué que la connectivité dans beaucoup de sortes de réseaux tels que le WWW, montre une distribution des degrés qui suit une loi de puissance et ce, indépendamment de l'échelle considérée. Cette propriété n'avait pas encore été rencontrée dans les modèles théoriques existants et ils ont donc proposé le modèle du réseau indépendant de l'échelle (Barabási, Albert et Jeong, 2000). L'équipe de Barabási a montré que les modèles de réseaux existants ne tenaient pas compte de deux caractéristiques importantes de certains réseaux réels. Premièrement, certains réseaux grandissent continuellement par l'addition constante de nouveaux sommets et deuxièmement, les nouveaux sommets se connectent préférablement à des sommets qui ont une connectivité élevée (attachement préférentiel). Ils ont démontré que la combinaison de la constante croissance du réseau et de l'attachement préférentiel est directement responsable de la distribution de la connectivité en loi de puissance. Ce modèle dynamique est défini par deux règles :

1. **Croissance** : en débutant avec un petit nombre de sommets m_0, à chaque pas de temps, il ajoute un nouveau sommet avec m nouveaux liens (où $m \leq m_0$) qui seront connectés à m sommets déjà présents dans le graphe.

2. **Attachement préférentiel** : Le choix des m sommets auxquels le nouveau sommet sera connecté s'effectue selon une probabilité p qui dépend de la connectivité du sommet potentiel : $p(k_i) = k_i / \sum_j k_j$

La figure 2.4.1.6 illustre ce type de réseau. On y remarque que certains sommets (les trois sommets noirs) possèdent une connectivité beaucoup plus élevée que les autres sommets à cause des mécanismes d'attachement préférentiel.

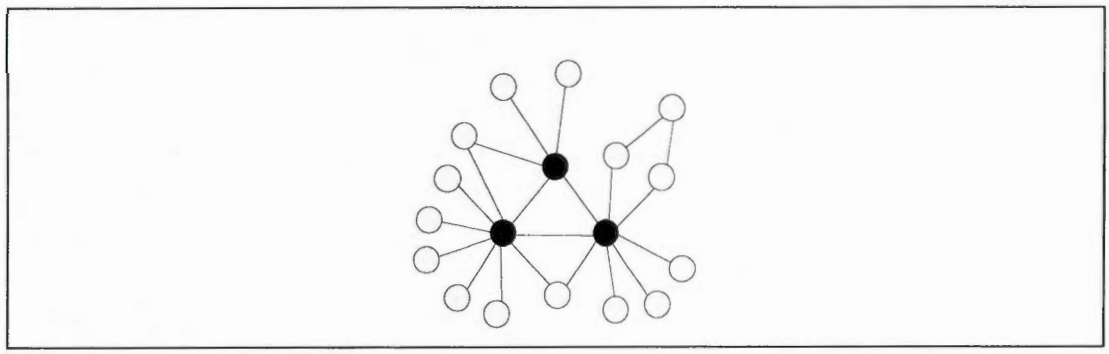


Figure 2.4.1.6 Modèle du réseau indépendant de l'échelle illustrant l'attachement préférentiel.

Partant de ce modèle, (Albert et Barabási, 2000) proposent une dynamique de reconnexion et de suppression de liens dans le temps. Dans le même optique, (Dorogovtsev et Mendes 2000) suggèrent un modèle similaire mais qui intègre l'idée que la tendance d'un site à attirer de nouveaux liens diminue selon son ancienneté : plus un site est âgé, moins il attire de nouveaux liens. Finalement, (Amaral et al., 2000) proposent un coût ou une contrainte de capacité qui ralentit le processus de la distribution des degrés en loi de puissance.

Un modèle de réseau d'amis

(Jin, Girvan et Newman, 2001) proposent un modèle représentant l'évolution d'un réseau social de contacts personnels (ou d'amis). Les propriétés de ce modèle diffèrent des règles du modèle de l'évolution du WWW (modèle indépendant de l'échelle et ses descendants) pour plusieurs raisons.

Tout d'abord, dans un réseau social, il est vrai que de nouveaux contacts et de nouveaux liens s'ajoutent et disparaissent continuellement dans le réseau. Des individus voyagent et changent de réseau, ils font de nouvelles connaissances, certains naissent, d'autres meurent, etc. Étant donné ce va-et-vient constant dans les réseaux sociaux, les auteurs de ce modèle considèrent que l'ajout et la perte de sommets s'équilibreront, en quelque sorte, et qu'il est raisonnable de considérer comme une bonne approximation que, dans l'ensemble, le nombre de sommets du réseau social modélisé ne varie pas. Seuls le nombre et l'arrangement des liens changent.

Deuxièmement, la distribution des degrés, dans un réseau social, ne semble pas suivre une loi de puissance, mais semble plutôt tourner autour d'un degré moyen. Ce phénomène s'explique par le fait que, dans la plupart des réseaux sociaux (certaines exceptions existent par exemple, les réseaux de contacts sexuels), l'entretien d'une amitié demande des ressources en temps et en efforts : il y a une limite au nombre de contacts qu'un individu peut entretenir.

Ensuite, l'absence d'une distribution des degrés suivant une loi de puissance suggère que le mécanisme de l'attachement préférentiel n'est pas très important dans les réseaux d'amis.

Finalement, comme on l'a vu précédemment, les réseaux sociaux doivent tenir compte de la propriété de transitivité qui, cependant, est absente du modèle du réseau du Web.

Étant donné ce qui vient d'être discuté, les auteurs proposent quatre caractéristiques importantes qu'un modèle dynamique de l'évolution d'un réseau d'amis devrait considérer :

1. **Un nombre fixe de sommets** : Considération d'une population fermée d'une grandeur fixe.

2. **Un degré limité** : La probabilité qu'un individu développe une nouvelle amitié doit diminuer fortement lorsque le nombre de ses amis atteint un certain niveau.

   Mécanisme mis en place pour modéliser cette propriété :

   z^* = limite sur le nombre z d'amis qu'un individu peut avoir. À l'atteinte de z^* contacts, pour un individu, la probabilité que cet individu crée d'autres liens d'amitié diminue rapidement.

3. **Transitivité** : La probabilité que deux individus deviennent des amis doit être significativement plus élevée s'ils ont un ou plusieurs amis communs.

   Mécanisme mis en place pour modéliser cette propriété :

   Des paires d'individus se rencontrent avec une probabilité par unité de temps qui dépend du nombre d'amis qu'ils ont en commun.

   La probabilité par unité de temps p_{ij} que deux personnes données, i et j, se rencontrent dépend de deux facteurs :

   a. Le nombre d'amis z_i et z_j que i et j possèdent déjà.

   b. Le nombre m_{ij} d'amis communs de i et j.

   Ces facteurs sont représentés par les fonctions f et g :

   $$p_{ij} = f(z_i)f(z_j)g(m_{ij})$$

   La fonction f doit être assez grande et assez constante pour de petits z, mais doit diminuer considérablement lorsque z approche de la valeur de transition z^*. La fonction de Fermi présente ces caractéristiques et c'est elle que les auteurs ont utilisée pour leur modèle.

$f(z_i) = \frac{1}{e^{\beta(z_i - z^*)} + 1}$ où le paramètre β contrôle la précision de la diminution à z^*.

La fonction *g(m)* représente l'augmentation attendue de la chance que deux individus se rencontrent s'ils ont un ou plusieurs amis communs.

$$g(m) = 1 - (1 - p_0)e^{-\alpha m}$$

Où p_0 représente la probabilité d'une rencontre entre deux personnes qui n'ont pas d'amis communs et où le paramètre α contrôle la vitesse à laquelle $g(m_{ij})$ augmente.

4. **Bris de liens d'amitié** : Étant donné le nombre fixe de sommets dans le réseau et le degré limité pour chaque sommet, il est nécessaire que des liens se brisent et que d'autres se forment pour que le réseau ne stagne pas.

   Mécanisme mis en place pour modéliser cette propriété :

   Lorsque deux individus sont amis, ceux-ci doivent se rencontrer régulièrement pour maintenir leur amitié. Si tel n'est pas le cas, il y a cessation de cette amitié.

   Pour représenter la force d'une amitié, chaque lien d'amitié possède un paramètre *s* dont la valeur est plus ou moins élevée selon que les rencontres sont plus ou moins fréquentes.

   Chaque fois que deux amis se rencontrent, la force s_{ij} est ajustée à la valeur 1. Puis, à mesure que le temps passe et qu'ils ne se rencontrent pas de nouveau, la force de leur amitié diminue exponentiellement selon :

   $$s_{ij} = e^{-k \Delta t}$$

   Où Δt représente l'intervalle de temps depuis la dernière rencontre de *i* et *j* et où *k* est un paramètre ajustable du modèle. Si *i* et *j* se rencontre une autre fois, s_{ij} est remis à 1. Il

est ensuite possible de poser une limite minimale à s_{ij} afin de considérer qu'en dessous de cette limite, il y a cessation de l'amitié entre i et j.

2.4.2 Modèles statistiques de réseaux sociaux

Dans l'approche sociologique, une des perspectives dominantes de l'analyse de réseaux voit les caractéristiques du réseau comme des variables pouvant expliquer le comportement de chaque individu au sein de l'ensemble social (les comportements qu'ils adoptent, les stratégies qu'ils utilisent) et ceci, en regard de la position que chacun d'eux occupe dans la structure du réseau. Dans ce cas, on ne cherche pas à expliquer la structure même du réseau.

Une seconde perspective, plus récente, s'intéresse plutôt aux relations. Elle tente d'expliquer leur émergence, leur agencement, leur évolution. De ce point de vue, la structure des réseaux doit être expliquée à partir du comportement social des acteurs qui le constituent.

En comparaison avec les modèles décrits plus haut, plusieurs sociologues ont choisi de construire des modèles statistiques. Nous allons nous concentrer ici sur quatre classes de modèles importants dans ce domaine qui ont donné naissance à plusieurs descendants : le modèle de Holland-Leinhardt et ses descendants, les modèles métriques, les modèles de fermeture des triades et finalement, les modèles de la variance du degré. Ceux-ci ont en commun qu'étant donné un couple d'individus et de leurs caractéristiques, ils se concentrent sur la probabilité de l'existence de relation entre eux. Il est à remarquer que la majorité de ces modèles gèrent des relations orientées ce qui n'est pas la tendance observée dans les modèles théoriques précédents.

Le modèle de Holland-Leinhardt et ses descendants

Le modèle *p1* (premier modèle de probabilité) est un modèle log-linéaire qui a été développé par (Holland et Leinhardt, 1981). Ce modèle permet d'une part, de spécifier des effets individuels en permettant de tenir compte, par exemple, de la popularité des acteurs (nombre de liens reçus = demi-degré intérieur) et de l'activité de ceux-ci (relations émises ou demi-

degré extérieur). Cependant, ce modèle ne permet pas de tenir compte de variables exogènes telles que l'âge, le sexe, le statut social qui sont souvent très importantes pour le développement de la structure d'un réseau relationnel. D'autre part, ce modèle permet de représenter des effets dyadiques (caractéristiques des acteurs et de leurs relations deux à deux) de réciprocité (probabilité que si *i* émet une relation vers *j* alors *j* émet aussi une relation vers *i*). Bien qu'étant un point de départ important pour le développement des modèles statistiques de réseaux sociaux, ce modèle postule l'indépendance entre les dyades, ne permet pas de représenter les effets triadiques et en ce sens, empêche le traitement au niveau structural.

Le modèle *p2*, développé par (Van Duijn et Snijders, 1995) est une amélioration du modèle *p1* en ce sens qu'il permet le traitement de variables exogènes (caractéristiques des acteurs) et de variables dyadiques (similarités, flux, fréquences, contenus ou toutes autres caractéristiques des relations). Cependant, ce modèle ne traite toujours pas les effets au niveau des triades.

En plus de posséder tous les avantages du modèle *p2*, le modèle *p**, introduit par (Wasserman et Pattison, 1996), propose une formulation très générale qui permet la représentation de n'importe quelles distributions de probabilités pour un graphe orienté. Ceci rend ce modèle très souple et permet le traitement des effets triadiques et donc structuraux. Cependant, la formulation est tellement générale qu'il faut trouver la formule spécifique pour chaque cas particulier et ceci n'est pas toujours évident.

Les modèles métriques

La méthode principale utilisée ici est la construction d'un espace social où chaque agent est localisé. C'est donc une représentation spatiale, et non purement structurale. Un algorithme stochastique est alors utilisé pour créer les liens sociaux étant donné une fonction de proximité. En général, cette fonction est la distance euclidienne entre chaque couple d'individus sélectionné. Plusieurs travaux existants retiennent cette solution soit en créant des réseaux sociaux qui dépendent seulement d'une distance géographique (Nowak et

Valacher, 1998) ou en incluant d'autres attributs tels que des indices socio-économiques de chaque individu. Une généralisation de la construction d'une métrique sociale a été proposée par (Banks et Carley, 1994). Il est à noter que le concept répandu parmi plusieurs de ces modèles est le concept de l'homophilie. Ce terme réfère à la tendance de la similarité sur plusieurs attributs des personnes qui se lient : plus les agents sont semblables ou plus ils sont près les uns des autres à l'intérieur d'un espace social, plus grande est leur tendance à être en relation. Une version spatiale de la construction d'un espace social est proposée par (Epstein et Axtell, 1996) dans laquelle les agents se déplacent sur une carte et les individus qu'ils rencontrent sont aussitôt intégrés à leurs réseaux sociaux. Les liens sont brisés lorsque les individus deviennent trop éloignés les uns des autres.

Les modèles de fermeture des triades (*completion triad*)

Ces modèles sont dérivés des travaux de (Heider, 1958) sur l'équilibre. Dans ces modèles, les individus cherchent à minimiser le déséquilibre perçu. Par exemple, si Louis se considère comme un ami de Lucie et que Louis perçoit que Lucie est une amie de Fabien alors, si Louis n'est pas ami avec Fabien, il percevra un déséquilibre et aura tendance à vouloir se lier d'amitié avec Fabien. L'objet principal n'est alors plus seulement la dyade, mais la triade, formée par un groupe de trois personnes et des relations entre elles. Le but de ce modèle est d'assigner une probabilité sur la création d'un nouveau lien à l'intérieur d'une triade, étant donnés les liens existants. Ceci a donné lieu à une série de modèles incluant le modèle de « clustering » de (Davis, 1967), le modèle de transitivité de (Holland et Leinhardt, 1971) et le modèle de l'équilibre positif de Newcomb (Newcomb, 1968).

Les modèles de la variance du degré (degree variance model)

Ces modèles dérivent en partie des travaux de (Blau, 1967) sur la théorie de l'échange. Ces modèles assument que chaque acteur est différent dans sa probabilité intrinsèque d'attirer de nouveaux liens : les acteurs qui ont un degré de centralité élevé tendent à attirer plusieurs liens tandis que ceux qui ont un degré de centralité faible tendent à en attirer moins. On dit d'un acteur qu'il est très central lorsqu'il est engagé dans beaucoup de relations (directes ou

indirectes). Plus le degré de centralité est élevé, plus l'acteur possède du pouvoir, plus il est populaire et plus il attire de nouvelles relations. Ce phénomène s'apparente fortement à celui de l'attachement préférentiel exposé dans le modèle du réseau indépendant de l'échelle (Barabási, Albert et Jeong, 2000). La polarisation et la balkanisation suivent aussi des conceptions théoriques du pouvoir et peuvent être vues comme des variations du modèle de la variance du degré. La polarisation se produit lorsque la société se divise en deux groupes, chacun étant centré autour d'un acteur spécifique, comme il se produit lorsqu'il y a des cliques opposées, chacun avec un puissant leader. La balkanisation se produit lorsque la société se divise en un grand nombre de petits groupes, chacun étant centré autour d'un acteur principal.

2.4.3 Approche par modélisation ascendante

Dans les approches basées agents, le réseau peut être vu comme un ensemble de relations générées par les agents du système. Il est nécessaire, alors, de définir les mécanismes de création, d'évolution et de suppression des relations des agents eux-mêmes.

Cette catégorie de modèles de réseaux sociaux utilise une approche de modélisation ascendante. Son but concerne la modélisation des mécanismes individuels sociocognitifs utilisés pour bâtir, entretenir et terminer des relations. On différencie deux approches. La première est issue de la théorie des jeux qui est axée sur les mécanismes rationnels de création de liens. La seconde, fortement liée avec les théories sociopsychologiques, privilégie la modélisation des propriétés sociocognitives de la création de relations. Étant donnés ces mécanismes, elle a pour but d'observer le genre de réseau qu'ils engendrent.

Modèles basés sur la théorie des jeux

En économie et dans la théorie des jeux en particulier, le cadre utilisé est celui de la maximisation de la fonction d'utilité. Chaque individu possède la même fonction utilitaire, mais peut posséder différentes ressources pour investir dans des relations ou pour modifier son comportement. Ces modèles s'intéressent donc à la création des relations et à leur

évolution (Myerson, 1977). La création ou l'entretien de chaque relation a un coût pour l'agent et les possibilités de partage des gains dépendent de la structure de son réseau social personnel (Slikker et Van Den Nouweland, 2001). Ainsi, l'agent doit bien choisir ses relations parce qu'elles sont coûteuses, mais nécessaires pour l'acquisition de ressources (Bala et Goyal, 1999). Le but de ces modèles est d'étudier le dilemme entre la stabilité et l'efficacité des réseaux générés (Jackson, 2001). Selon cette approche basée sur la fonction d'utilité, (Stokman et Van Oosten, 1994 ; Stokman et Zeggelink, 1996) ont développé des modèles de la négociation politique. Dans ce cas, on étudie la dynamique des interactions entre les agents plutôt que la structure de leurs interactions. Cependant, une autre tendance dans la théorie des jeux est d'appliquer des jeux classiques et de les tester dans différentes structures sociales où seulement les agents reliés peuvent interagir. Dans ce cas, les économistes utilisent souvent des modèles théoriques de réseaux sociaux tels que les graphes aléatoires, les structures régulières ou les petits mondes comme structure d'interaction (Peyton Young, 1998).

Simulations sociales basées agents

Le but de la modélisation basée agents dans l'étude des réseaux sociaux est d'exprimer des mécanismes sociocognitifs de création, d'entretien et de finalisation de relations. La base théorique de ces mécanismes vient surtout des théories sociopsychologiques du comportement de groupe. Ces modèles initialisent le réseau social en choisissant parfois un réseau vide ou bien en utilisant un réseau choisi parmi les modèles théoriques ou encore, en utilisant des données empiriques. Les modèles basés agents sont nombreux et plus ou moins complexes. Pour n'en nommer que quelques-uns, (Mosler et Brucks, 2001) justifient l'utilisation de simulations basées agents pour explorer des théories sociopsychologiques, (Flache et Macy, 1996) utilisent une approche stochastique d'apprentissage basée agents pour explorer la théorie de l'échange de Homan, (Zeggelink, 1993) suggère de nouveaux modèles de réseaux d'amitié basés agents et (Snijders, 1996) propose des modèles stochastiques orientés acteurs pour tester la théorie de la fraternité de Newcomb.

Dans un modèle basé agents, ce qui motive un agent à créer et maintenir des relations peut être, par exemple, son désir d'atteindre certains buts ou d'occuper une position stratégique dans le réseau qui augmentera son pouvoir. Aussi, dans plusieurs modèles, la notion de confiance joue souvent un rôle important dans le choix des relations d'un agent.

2.5 Outils de modélisation et simulation

Il existe une panoplie d'outils, de logiciels et de librairies qui peuvent aider à la programmation d'un modèle de réseau social dans le but de le simuler. L'objectif de cette section est de positionner NetSim par rapport aux différents types d'outils déjà à notre disposition.

Tout d'abord, on constate qu'il existe plusieurs langages ou logiciels qui sont bien adaptés pour la mise en œuvre de simulations d'ensembles sociaux complexes. Ces outils permettent de définir finement les agents de la simulation, les mécanismes d'interactions entre les agents et l'environnement dans lequel évoluent ces agents. Nous parlons ici d'applications capables de construire des modèles multi-agents très complexes. *Swarm* (http://www.swarm.org), par exemple, est une des meilleures plates-formes pour la création de simulations sociales. Elle a été spécialement conçue pour la simulation d'agents multiples dans des systèmes adaptatifs complexes. *Ascape* (http://www.brook.edu/es/dynamics/ models/ascape) et *RePast* (http://repast.sourceforge.net) sont d'autres exemples de ce genre de logiciels.

Étant donné leur puissance et leur flexibilité, ces applications permettent d'implémenter des mécanismes sociaux très sophistiqués. Par contre, il demande une longue courbe d'apprentissage et beaucoup de programmation. Dans le cas de *Swarm*, par exemple, il est nécessaire d'avoir une certaine expérience en Java ou en Objective C, d'être familier avec l'orienté-objet et d'apprendre le code Swarm.

Ces logiciels ne sont pas comparables à NetSim. Ils sont destinés à simuler des modèles plutôt basés agents qui peuvent modéliser des comportements sociaux très complexes. De

plus, ils visent plus particulièrement l'étude de la dynamique des interactions entre les agents plutôt que de s'intéresser, comme NetSim, à la structure de ces interactions.

D'autre part, en ce qui concerne plus particulièrement l'étude de la structure des réseaux, on peut trouver une grande variété de logiciels et librairies qui permettent la représentation et l'analyse de données de réseaux. Ils comprennent des méthodes pouvant calculer les notions les plus courantes comme la densité, le coefficient de transitivité, la distribution des degrés, la centralité, l'équivalence structurale et parfois, des procédures d'analyse statistique plus sophistiquées. Les logiciels

- UCINET (http://www.analytictech.com/downloaduc6.htm) et
- StOCNET (http://stat.gamma.rug.nl/stocnet),

en sont des exemples.

D'autres outils se spécialisent dans la visualisation, sous forme de graphe, de données de réseaux. Ils sont souvent offerts sous forme de librairie comme

- JUNG (http://jung.sourceforge.net),
- GINY (http://csbi.sourceforge.net) et
- Prefuse (http://www.prefuse.org)

ou sont des applications comme

- Graphviz (http://www.graphviz.org/),
- KrackPlot (http://www.isi.edu/~blythe/KP) ou
- NetDraw (http://www.analytictech.com/downloadnd.htm).

Ces différents outils sont très utiles pour étudier la structure de réseaux statiques, mais ne permettent pas de spécifier les règles d'un modèle dynamique afin de le simuler dans le temps. En général, les simulations sont programmées pour l'étude d'un phénomène

particulier et ne servent que pour cette étude. À notre connaissance, une plate-forme générique comme NetSim, permettant de modéliser les règles de divers modèles de réseaux sociaux à l'aide d'un langage de modélisation spécifique, et ce, dans le but de les simuler et de visualiser l'évolution de leurs structures dans le temps, n'a pas encore été développée. Nous voudrions cependant mentionner *Blanche* (Hyatt, Contractor et Jones, 1997), un outil informatique servant à la modélisation et à l'exécution de simulations numériques basées sur la conceptualisation et l'organisation des réseaux. Un peu comme NetSim, il fournit une architecture flexible et réutilisable pour la spécification de divers types de modèles en permettant la définition de différentes entités (acteurs, personnes, noeuds), d'attributs sur ces entités, de relations entre ces entités et d'équations mathématiques qui expriment les règles du modèle à simuler. Cependant, contrairement à NetSim, cette application ne permet pas la visualisation, sous forme de graphe, de l'évolution structurale du réseau au cours de la simulation. Par contre, elle fournit différentes méthodes d'analyse du réseau de sortie de la simulation.

2.6 Conclusion

Tout d'abord, nous avons vu, dans ce chapitre, quelques exemples montrant la pertinence de l'étude des réseaux sociaux dans plusieurs domaines puis nous avons expliqué diverses méthodes d'analyse de réseaux, provenant de la sociologie structurale, qui sont très utiles à la compréhension de certains phénomènes sociologiques. Nous avons vu différentes méthodes de partitionnement de réseaux pour une étude de la structure au niveau global et avons ensuite expliqué des procédures de positionnement des acteurs dans le réseau pour une étude au niveau local.

Le logiciel NetSim n'a pas encore développé de méthode d'analyse des réseaux, mais nous pensons tout de même que la compréhension de ces concepts est pertinente pour effectuer une première analyse visuelle de la structure des réseaux engendrés par simulation. Les concepts de centralité, de prestige, par exemple, bien que pouvant se calculer de façon mathématique, peuvent être facilement repérables, dans certains cas, par simple visualisation.

Nous avons ensuite montré que l'investigation de certains phénomènes sociologiques peut se faire par modélisation et avons décrit plusieurs modèles provenant de diverses disciplines. Premièrement, les modèles théoriques, provenant surtout de la physique et des mathématiques, tentent de représenter des phénomènes observés dans la réalité tels que la transitivité et l'effet des petits mondes dans plusieurs types de réseaux sociaux et l'attachement préférentiel, dans l'évolution, par exemple, de la structure du Web. Deuxièmement, les modèles statistiques, utilisés surtout en sociologie, essaient de comprendre et d'expliquer la structure des réseaux sociaux par les relations qu'entretiennent les acteurs entre eux. Ces modèles étudient la probabilité de l'existence d'une relation entre deux acteurs selon leurs caractéristiques et leur position dans le réseau. Finalement, nous avons vu l'approche par modélisation ascendante qui modélise les mécanismes individuels de création, d'entretien et de suppression de relations des agents eux-mêmes dans le but d'étudier le genre de réseau qu'ils engendrent.

Finalement, nous avons positionné NetSim par rapport à divers types d'outils de modélisation, de visualisation et de simulation de réseaux.

CHAPITRE 3

Langage de modélisation NetSim

3.1 Introduction

Les chapitres précédents présentaient le contexte théorique et certaines notions de base qui ont motivé la réalisation de ce projet. Nous avons constaté qu'il existe des modèles dynamiques de réseaux d'information qui peuvent être étudiés par simulation. Nous avons vu que les structures des réseaux ainsi générés peuvent être examinées par des méthodes d'analyse, basées sur la théorie de graphes, dont les principaux concepts sont souvent repérables par simple visualisation graphique.

De plus, nous avons vu qu'il existe beaucoup de ressources (logiciels, librairies etc.) qui aident à la modélisation, la simulation, la visualisation et l'analyse des réseaux d'information. Cependant, on constate que, pour plusieurs de ces outils, la mise en œuvre d'une simulation peut être assez laborieuse et demande souvent de bonnes compétences en programmation. De plus, chaque fois que l'on désire implémenter un nouveau modèle, le travail est souvent à recommencer. Ces constats nous ont amenés à chercher des solutions pour faciliter la mise en oeuvre de simulations diverses.

C'est dans cet esprit, donc, que nous avons développé une plate-forme générique de modélisation, de simulation et de visualisation de modèles dynamiques de réseaux d'information.

Dans ce chapitre, nous allons aborder plus particulièrement la partie modélisation qui consiste en un langage de modélisation permettant de spécifier les paramètres et les règles des modèles que l'on veut simuler. Ce langage est, en quelque sorte, le concept central de ce

projet et a été élaboré en particulier pour les modèles de réseaux sociaux et le Web. Nous croyons cependant que sa flexibilité et son extensibilité le rendent capable d'exprimer divers types de réseaux d'information.

Les modèles qui nous intéressent plus particulièrement sont les modèles dynamiques pouvant, de ce fait, être simulés par ordinateur et dont la structure d'interactions sous-jacente est un graphe. Nous en avons vu quelques-uns au chapitre précédent et allons les utiliser, dans les chapitres à venir, à titre d'exemple pour la modélisation avec le langage NetSim.

3.2 Les entités du langage

Le langage de modélisation NetSim a été élaboré en observant différents modèles dynamiques de réseaux d'information (Jin, Girvan et Newman, 2001 ; Albert et Barabási, 2000). Les entités manipulées dans ces modèles sont à peu près toujours les mêmes : les acteurs ou agents, les relations entre les acteurs et finalement, l'ensemble des acteurs et de leurs relations qui forment le réseau. Dans le langage NetSim, le réseau est représenté par un graphe dans lequel les acteurs sont représentés par les nœuds ou sommets du graphe et les relations, par les liens (arcs ou arêtes) du graphe.

Le tableau 3.2.1 donne une description des entités du langage de NetSim. Pour l'instant, notre logiciel de simulation ne permet de simuler que l'évolution d'un seul réseau à la fois, mais nous pensons qu'éventuellement, il serait intéressant d'étendre les fonctionnalités de NetSim pour que celui-ci puisse exécuter plusieurs simulations différentes en même temps. Dans cette optique, nous avons prévu l'entité world qui représente l'environnement dans lequel évoluent les réseaux. Cet environnement pourrait alors devenir un lieu d'échange de données entre les différents réseaux. Pour le moment, puisqu'il n'y a qu'un seul réseau, cette entité est pratiquement triviale.

Tableau 3.2.1 Les entités du langage NetSim

Entité	Représentation
world	L'environnement dans lequel évolue le réseau d'information simulé.
network	Le réseau formé par les acteurs et les relations qu'ils entretiennent entre eux.
link	Une relation entre deux acteurs.
node	Un acteur

Certaines entités, comme nous le verrons dans les sections suivantes, possèdent des constantes, des attributs et des fonctions prédéfinis par NetSim ou qui peuvent être définis par l'utilisateur. Les constantes sont initialisées au début de la simulation et leur valeur reste inchangée durant toute la simulation. Les attributs, par contre, sont initialisés au début de la simulation, mais leur valeur peut être modifiée au cours de celle-ci. Les fonctions, pour leur part, sont des valeurs calculées pendant la simulation.

Pour comprendre ce langage de modélisation, on doit tout d'abord comprendre la hiérarchie des entités qui détermine la visibilité d'une entité par une autre :

1. Les attributs et fonctions définis sur l'entité world sont visibles par toutes les entités.
2. Les attributs et fonctions définis sur l'entité network sont visibles par les entités network, link et node.
3. Les attributs et fonctions définis sur l'entité link ne sont visibles que par l'entité link seulement.
4. Les attributs et fonctions définis sur l'entité node ne sont visibles que par l'entité node seulement.
5. Les constantes sont visibles de partout.

La figure 3.2.1 illustre cette hiérarchie de façon plus schématique.

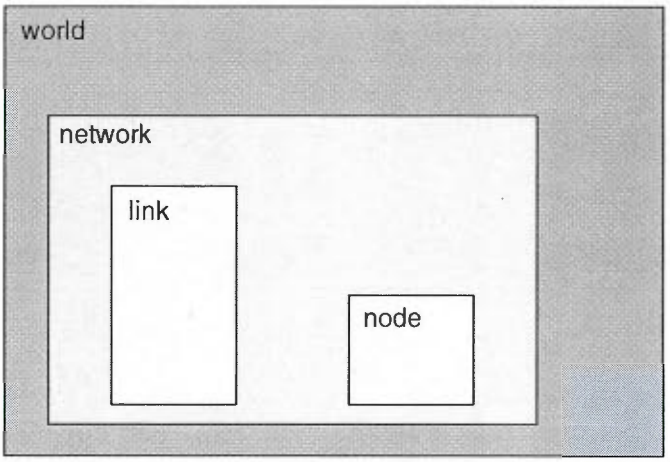


Figure 3.2.1 Hiérarchie des entités de NetSim
Chaque entité est visible par elle-même et par les entités représentées par une teinte de gris plus pâle (s'il y a lieu). Dit d'une autre manière, chaque entité peut se voir elle-même et peut voir des entités représentées par une teinte de gris plus foncée. Par exemple, l'entité world ne voit qu'elle-même, l'entité network peut se voir elle-même ainsi que l'entité world. L'entité link peut se voir elle-même ainsi que l'entité network et l'entité world et finalement, l'entité node peut se voir elle-même ainsi que l'entité network et l'entité world.

La section suivante explique la syntaxe du langage NetSim pour écrire ce que nous appelons les définitions du modèle : les constantes, les attributs et les fonctions.

3.3 Syntaxe d'écriture des définitions du modèle

3.3.1 Généralités

Le langage NetSim permet de définir certains paramètres sur les entités de la simulation. Ces paramètres (définitions) peuvent être des constantes, des attributs ou des fonctions.

À l'exception des constantes qui sont visibles de partout, chaque définition est définie sur une entité spécifique (world, network, link ou node) qui détermine sa visibilité comme nous l'avons vu à la section précédente.

Les commandes servant à écrire les définitions commencent toutes par le mot « DEFINE ».

Chaque définition se termine par un point-virgule.

3.3.2 Les constantes

Les constantes sont des valeurs numériques déterminées au début de la simulation et qui sont utilisées dans la spécification des règles du modèle. Celles-ci ne sont pas modifiables pendant la simulation.

La commande pour définir une constante est `DEFINE_CONST` et prend comme paramètre le nom de la constante ainsi que sa valeur d'initialisation.

Par exemple, pour définir la constante dont le nom est « limite » et dont la valeur d'initialisation est 5, il faut écrire :

```
DEFINE_CONST (limite, 5);
```

3.3.3 Les attributs

Les attributs sont des variables définies sur une entité particulière. Leur valeur est initialisée au début de la simulation, mais peut être modifiée au cours de la simulation. NetSim définit quelques attributs sur les entités node et link qui caractérisent leur apparence lors de la visualisation de la simulation cependant, des attributs supplémentaires peuvent être définis par l'utilisateur.

3.3.3.1 Les attributs prédéfinis par NetSim

Les entités node et link possèdent une série d'attributs de visualisation prédéfinis qui ne concernent que les propriétés graphiques de ces entités. On parle ici de grosseur et/ou de couleur des nœuds ou des liens. En effet, il peut être utile, par exemple, de colorer certains nœuds ou liens présentant des caractéristiques particulières que l'on voudrait mettre en évidence en cours de simulation. Par exemple, on pourrait vouloir colorer en vert les nœuds qui ont un degré plus élevé que la moyenne afin de les rendre plus visibles en cours de

simulation. Les tableaux 3.3.3.1.1 et 3.3.3.1.2 montrent la liste de ces attributs pour chaque entité.

Tableau 3.3.3.1.1 Attributs prédéfinis sur l'entité node

Attributs	Description
border_color	Couleur de la bordure du nœud (noir par défaut).
fill_color	Couleur de remplissage du nœud (noir par défaut).
highlight_color	Couleur de remplissage du noeud lorsque celui-ci est mis en évidence par la souris : lorsque la souris est positionnée sur un nœud, ce nœud ainsi que tous ses voisins prennent cette couleur (rouge par défaut).
fixed_color	Couleur de remplissage du nœud lorsque celui-ci est immobile dans la visualisation graphique : il est possible d'immobiliser un nœud en cliquant dessus avec la souris (rouge par défaut).
radius	Grosseur des nœuds. Plus cette valeur numérique est petite, plus la grosseur des nœuds est petite et vice versa (égal 6 par défaut).

Tableau 3.3.3.1.2 Attributs prédéfinis sur l'entité link

Attributs	Description
color	La couleur du lien (noir par défaut).
highlight_color	La couleur du lien lorsque celui-ci est mis en évidence par la souris : lorsque la souris est positionnée sur un nœud, tous les liens de ce nœud prennent cette couleur (rouge par défaut).

En ce qui concerne les attributs de couleur, ce sont des valeurs numériques qui correspondent à des codes de couleur spécifiés dans un fichier de configuration de NetSim.

3.3.3.2 Définition d'attributs par l'utilisateur

Les attributs prédéfinis par NetSim sont des attributs de visualisation et ne concernent que les entités link et node. Cependant, les attributs servent souvent à représenter l'état de chaque entité au cours d'une simulation. Le langage de modélisation NetSim offre donc la possibilité à l'utilisateur de définir des attributs d'état sur toutes les entités de la simulation. Les attributs sont toujours numériques et peuvent être modifiés au cours de la simulation.

La commande de définition des attributs est `DEFINE_ATTRIB` et prend comme paramètres l'entité sur laquelle on définit l'attribut (world, network, link ou node), le nom de l'attribut ainsi que sa valeur d'initialisation.

Par exemple, nous pourrions définir, sur l'entité link, l'attribut dont le nom est « force » et dont la valeur d'initialisation est 1 de cette manière :

```
DEFINE_ATTRIB (link, force, 1);
```

La définition ci-dessus implique que chaque lien nouvellement créé au cours de la simulation possèdera alors un attribut « force » initialisé à 1 et dont la valeur pourra, bien entendu, être modifiée par la suite. Selon les règles de visibilité vues à la section 3.2, cet attribut ne sera visible que par l'entité link. De la même façon, nous pourrions définir des attributs sur les entités world, network et node.

3.3.4 Les fonctions

Après avoir écrit les constantes et les attributs du modèle dans le langage de modélisation, on peut alors écrire les fonctions nécessaires au fonctionnement de notre modèle. Les fonctions NetSim sont des expressions arithmétiques ou booléennes qui calculent et retournent une valeur numérique ou booléenne. Comme pour les attributs, le langage fournit déjà certaines fonctions prédéfinies sur les entités, mais il est aussi possible, pour l'utilisateur, de définir ses propres fonctions. Dans ce dernier cas, nous aurons besoin de quelques opérateurs

arithmétiques et booléens. Les tableaux 3.3.4.1 à 3.3.4.5 montrent la liste des opérateurs de base dont dispose ce langage de modélisation.

Tableau 3.3.4.1 Opérateurs arithmétiques binaires de NetSim

Opérateurs arithmétiques binaires		
Signes	**Opérations**	**Exemples**
+	addition	5 + 3 = 8
-	soustraction	5 – 3 = 2
*	multiplication	5 * 3 = 15
/	division (réelle)	5 / 3 = 1.666…
**	puissance	5 ** 3 = 125
mod	modulo	5 **mod** 3 = 2

Tableau 3.3.4.2 Opérateurs arithmétiques unaires de NetSim

Opérateurs arithmétiques unaires		
Signes	**Opérations**	**Exemples**
-	nombre négatif	-1
+	nombre positif	+1 = 1
floor	plancher	**floor** 3.4 = 3
ceil	plafond	**ceil** 3.4 = 4

Tableau 3.3.4.3 Opérateurs booléens binaires de NetSim

Opérateurs booléens binaires		
Signes	**Opérations**	**Exemples**
=	test d'égalité	1 = 2 est faux ; 2 = 2 est vrai
/=	test d'inégalité	1 /= 2 est vrai ; 2 /= 2 est faux
<	plus petit	5 < 3 = faux ; 5 < 5 = faux
>	plus grand	5 > 3 = vrai ; 5 > 5 = faux
<=	plus petit ou égal	5 <= 3 = faux ; 5 <= 5 = vrai
>=	plus grand ou égal	5 >= 3 = vrai ; 5 >= 5 = vrai
and	et	
or	ou	
xor	ou exclusif	

Tableau 3.3.4.4 Opérateur booléen unaire de NetSim

Opérateur booléen unaire		
Signes	**Opérations**	**Exemples**
not	négation	**not** faux = vrai ; **not** vrai = faux

Tableau 3.3.4.5 Opérateur d'affectation

Opérateur d'affectation		
Signe	**Opération**	**Exemple**
:=	affectation	force := 5 (affecte la valeur 5 à l'attribut force)

3.3.4.1 Les fonctions prédéfinies par NetSim

Le langage de modélisation NetSim offre une série de fonctions définie sur les différentes entités. Une fonction se termine toujours par une parenthèse ouvrante et fermante pour se différencier des attributs.

Les tableaux 3.3.4.1.1 à 3.3.4.1.4 montrent les fonctions disponibles pour chaque entité. Pour ce travail, nous avons implémenté les fonctions de base mentionnées dans les tableaux suivants, mais nous souhaitons éventuellement ajouter de nouvelles fonctions pour enrichir notre langage. NetSim a été conçu de façon à être facilement extensible et nous verrons, au chapitre suivant, comment implémenter de nouvelles fonctions qui pourront ensuite être utilisées comme fonctions prédéfinies du langage de modélisation.

Tableau 3.3.4.1.1 Fonctions prédéfinies sur l'entité world

Fonctions prédéfinies sur l'entité world	
Selon les règles de visibilité des entités, cette fonction est visible par les entités world, network, link et node.	
Nom de la fonction	**Description**
time_step()	Retourne le nombre de pas de temps effectués depuis le début de la simulation. Le premier pas de temps = 0 et augmente de 1 à chaque pas de temps supplémentaire.
random_number()	Fonction utilitaire qui retourne un nombre aléatoire entre 0 et 1 inclusivement.

Tableau 3.3.4.1.2 Fonctions prédéfinies sur l'entité network

Fonctions prédéfinies sur l'entité network	
Selon les règles de visibilité des entités, ces fonctions sont donc visibles par les entités network, link et node.	
number_of_nodes()	Retourne le nombre de nœuds contenus dans le réseau.

number_of_links()	Retourne le nombre de liens contenus dans le réseau.
mean_degree()	Retourne le degré moyen du réseau : la somme des degrés de tous les nœuds du réseau divisée par le nombre de nœuds du réseau.
max_degree()	Retourne le degré d'un nœud du réseau qui est supérieur ou égal aux degrés de tous les autres nœuds.
min_degree()	Retourne le degré d'un nœud du réseau qui est inférieur ou égal aux degrés de tous les autres nœuds.
diameter()	Retourne le diamètre du réseau

Tableau 3.3.4.1.3 Fonctions prédéfinies sur l'entité link

Fonctions prédéfinies sur l'entité link	
Selon les règles de visibilité des entités, ces fonctions sont donc visibles par l'entité link uniquement.	
start_node()	Retourne le premier nœud du lien. Si le graphe est orienté, c'est le nœud qui correspond au point de départ du lien.
end_node()	Retourne le deuxième nœud du lien. Si le graphe est orienté, c'est le nœud qui correspond au point d'arrivée du lien.
number_of_common_contacts()	Retourne le nombre de voisins immédiats qu'ont en commun le premier nœud (de départ) et le second nœud (d'arrivée) du lien.
have_common_contacts()	Retourne vrai si les deux nœuds du lien ont au moins un contact en commun, retourne faux sinon.
a_common_contact()	Retourne un nœud qui est un contact commun des deux nœuds du lien.

Tableau 3.3.4.1.4 Fonctions prédéfinies sur l'entité node

Fonctions prédéfinies sur l'entité node	
Selon les règles de visibilité des entités, ces fonctions sont donc visibles par l'entité node uniquement.	
degree()	Retourne le degré du noeud. Pour un graphe orienté, c'est la somme du demi-degré extérieur et du demi-degré intérieur. Si le graphe est non orienté, le degré est identique au demi-degré intérieur et au demi-degré extérieur.
out_degree()	Retourne le nombre de liens sortant du nœud (demi-degré extérieur).
in_degree()	Retourne le nombre de liens entrant sur le nœud (demi-degré intérieur).

3.3.4.2 Définitions de fonctions par l'utilisateur

Outre les fonctions prédéfinies par le langage de modélisation NetSim, l'utilisateur peut définir ses propres fonctions. Une fonction définie par un utilisateur est une expression arithmétique ou booléenne qui retourne respectivement une valeur numérique ou booléenne. Comme dans le cas des attributs, une fonction est toujours définie sur une entité spécifique. Elle peut être construite à l'aide d'opérateurs arithmétiques ou booléens, de constantes, d'attributs et d'autres fonctions pourvu que les attributs et les autres fonctions respectent les règles de visibilité dictées par l'entité sur laquelle on définit la fonction.

La commande de définition des fonctions est `DEFINE_FUNCTION` et elle prend comme paramètres l'entité sur laquelle on veut définir la fonction et le nom de la fonction. Ensuite, on écrit l'expression de la fonction qui se termine par un point-virgule.

Lorsqu'on veut appeler une fonction, on doit écrire le nom de celle-ci suivi d'une parenthèse ouvrante et fermante. La fonction calcule et retourne la valeur de son expression au moment de l'appel. Supposons les définitions de constantes et d'attributs suivantes :

```
DEFINE_CONST (const_1, 2.34);
DEFINE_CONST (const_2, 0.1);
DEFINE_CONST (const_3, 0.8);

DEFINE_ATTRIB (node, node_1, 0);
DEFINE_ATTRIB (link, link_1, 0.000);
DEFINE_ATTRIB (network, net_1, 4);
DEFINE_ATTRIB (world, world_1, 30);
DEFINE_ATTRIB (world, world_2, 2);
```

Voici quelques exemples de définitions de fonctions :

Exemple 1 :

La fonction suivante est définie sur l'entité world et son nom est `f_1`. C'est une fonction booléenne, car elle teste l'égalité entre deux sous-expressions arithmétiques à l'aide de l'opérateur « = ». L'expression booléenne de cette fonction utilise la fonction `time_step()` et les attributs `world_1` et `world_2` aussi définis sur l'entité world. La visibilité des entités est donc respectée.

```
DEFINE_FUNCTION (world, f_1)
      time_step() ** world_2 = time_step() ** floor (world_1);
```

Exemple 2 :

La fonction suivante est définie sur l'entité network et son nom est `f_2`. C'est aussi une fonction booléenne, car elle utilise l'opérateur booléen « `>=` ». Elle utilise les fonctions `number_of_links()`, `number_of_nodes()` et `diameter()`, toutes prédéfinies sur l'entité network. Elle utilise aussi la fonction `time_step()` qui est prédéfinie sur l'entité world. Comme cette fonction est définie sur l'entité network, l'expression respecte bien les règles de visibilité, car l'entité world est visible par l'entité network.

```
DEFINE_FUNCTION (network, f_2)
      (number_of_links() + number_of_nodes()) / min_degree() >=
      diameter() * time_step();
```

Exemple 3 :

La fonction suivante est une fonction définie sur l'entité node et son nom est `f_3`. C'est aussi une fonction booléenne car elle utilise des opérateurs booléens. On remarque qu'elle fait appel aux fonctions `f_1()` et `f_2()` définies précédemment sur l'entité world et l'entité network. De plus, elle utilise les fonctions `in_degree()` et `out_degree()` prédéfinies sur l'entité node. L'expression respecte donc les règles de visibilité puisque les entités world et network sont visibles par l'entité node.

```
DEFINE_FUNCTION (node, f_3)
      not (f_1() or f_2() and (in_degree() > out_degree() or
      in_degree() /= out_degree()));
```

Exemple 4:

La fonction suivante est définie sur l'entité link et son nom est `f_4`. C'est une fonction arithmétique qui retourne une valeur numérique. On peut remarquer ici comment il est possible d'appeler une fonction définie sur node dans une fonction définie sur link. En effet, les deux fonctions `start_node()` et `end_node()`, définies sur link, retournent un nœud. Avec la notation pointée, il est possible d'appeler des fonctions ou d'utiliser des attributs définis sur l'entité node comme dans les expressions `start_node().degree()` et `end_node().node1`. Cette fonction respecte aussi les règles de visibilité.

```
DEFINE_FUNCTION (link, f_4)
      start_node().degree() / const_2 * ((end_node().node_1 +
      link_1) * const_3) / net_1;
```

Exemple 5 :

Cet exemple montre une fonction qui n'est pas bien définie. Elle présente deux erreurs. La première erreur vient de l'utilisation de l'attribut `link_1` dans une fonction définie sur node. En effet, l'entité link n'est pas visible par l'entité node. La seconde erreur vient de

l'utilisation de la fonction `f_1()` dans l'expression. En effet, cette fonction retourne une valeur booléenne qui ne peut pas être multipliée ainsi.

```
DEFINE_FUNCTION (node, f_4)
      degree() / const_2 * link_1 * f_1();
```

3.3.4.3 Définition de méthodes Java dans NetSim

NetSim est une application basée sur le langage Java et il est possible de définir des méthodes Java qui pourront, par la suite, être utilisées dans la spécification des règles du modèle à simuler. Étant donné qu'il est impossible de prévoir toutes les fonctions qui pourraient être nécessaires à la spécification des règles de tous les modèles possibles, cette commodité a été prévue dans le cas où NetSim n'offrirait pas la fonction prédéfinie nécessaire à la modélisation. Évidemment, l'utilisation de cette fonctionnalité suppose que l'utilisateur de NetSim connaît le langage Java, ce qui n'est pas toujours le cas. Éventuellement, nous aimerions implémenter plusieurs autres fonctions NetSim pour ainsi minimiser l'utilisation de méthodes Java.

Il y a cependant certaines restrictions à l'utilisation de méthodes Java dans NetSim. Premièrement, la méthode Java définie doit toujours retourner un nombre réel (Double). De plus, pour le moment, le résultat de cette méthode est toujours considéré comme une probabilité (un nombre entre 0 et 1 inclusivement). S'il advient que le résultat retourné est plus petit que zéro, il sera considéré comme égal à zéro. Si le résultat est plus grand que le nombre un, il sera considéré comme égal à un.

Pour pouvoir implémenter une méthode Java, il faut d'abord l'indiquer dans les définitions du modèle avec la commande `DEFINE_JAVA_METHOD`. Cette commande prend deux paramètres : l'entité sur laquelle est définie cette méthode et le nom de la méthode. Par exemple, pour définir la méthode `probabiliteAjouterCeLien` sur l'entité link, on écrit :

```
DEFINE_JAVA_METHOD (link, probabiliteAjouterCeLien );
```

Ensuite, il faut écrire le code Java de la méthode dans un espace réservé à cet effet dans l'interface graphique de NetSim. L'entête de la méthode Java correspondant à la définition ci-dessus doit être absolument :

```
public Double probabiliteAjouterCeLien (Link link);
```

Il faut mentionner ici que la classe dans laquelle est définie cette méthode contient déjà les objets nécessaires à la consultation ou la modification des attributs de chaque entité de la simulation. Nous avons donc la variable `world` qui représente l'entité world et la variable `network`, qui représente l'entité network. Dans le cas des entités link et node, lors de l'appel de cette méthode au cours de la simulation, l'instance de `Node` ou de `Link` qui est en train d'être traitée est passée en paramètre selon que la méthode est définie sur l'une ou l'autre de ces entités.

De plus, cette classe contient l'attribut `simController` qui est une instance de la classe `SimulationController` que nous verrons plus en détail au chapitre suivant. Il suffit de savoir, pour le moment, que toutes les méthodes servant à la consultation ou à la modification des attributs définis sur chaque entité du langage sont des méthodes définies sur cet objet. Par exemple, pour modifier la valeur de l'attribut `toto` qui aurait préalablement été défini sur l'entité world, nous écrivons :

```
simController.setAttribute(world, « toto », valeur);
```

où le paramètre `valeur` est un une variable de type double. Pour obtenir la valeur de l'attribut `force` qui aurait préalablement été défini sur l'entité link, nous écrivons :

```
Double valeur = simController.getAttribute (link, « force »);
```

Le `SimulationController` contient aussi l'équivalent Java de toutes les fonctions prédéfinies du langage de NetSim. Par exemple, pour obtenir le pas de temps courant de la simulation, le langage de modélisation NetSim offre la fonction `time_step()`. Pour obtenir cette valeur dans une méthode Java, il faut appeler la méthode `getTimeStep()` sur l'objet simController :

```
int t = simController.getTimeStep();
```

Finalement, le `SimulationController` contient évidemment d'autres méthodes de consultation et de gestion du graphe représentant le réseau simulé. Ces méthodes ont l'avantage, par rapport aux fonctions prédéfinies du langage de NetSim, de pouvoir retourner un résultat autre que numérique ou booléen. Nous ne ferons pas ici la liste exhaustive des méthodes disponibles dans le `SimulationController`, mais voyons tout de même quelques exemples : la méthode `getAllPotentialLinks()` retourne une liste (ArrayList) de tous les liens qui pourraient être ajoutés au réseau et la méthode `commonNeighbours (Node node1, Node node2)` retourne une liste de tous les voisins qu'ont en commun les paramètres `node1` et `node2`.

Nous n'aborderons pas cette fonctionnalité plus en détail, mais nous présentons un exemple simple de l'utilisation d'une méthode Java dans l'écriture d'un modèle de l'évolution du Web au chapitre 5.

3.3.5 Exemple d'écriture des définitions du modèle d'amitié de Jin, Girvan et Newman

Pour donner une idée plus concrète de l'écriture des définitions d'un modèle avec le langage de modélisation NetSim, nous allons traduire le modèle dynamique du réseau d'amis de (Jin, Girvan et Newman, 2001) présenté à la section 2.4.1.

Définitions des constantes du modèle

Dans l'exemple du modèle d'amitié, il y a plusieurs constantes que nous pourrions définir comme suit :

- z^* est le nombre d'amis après lequel la probabilité de créer d'autres amitiés diminue rapidement. Nous appelons cette constante `z_limite` et l'initialisons à 5 par exemple.

```
DEFINE_CONST (z_limite, 5);
```

- β contrôle la précision de la diminution à z^*. Nous appelons cette constante `b` et l'initialisons à 5 par exemple.

```
DEFINE_CONST (b, 5);
```

- α contrôle la vitesse de l'augmentation de la chance que deux individus se rencontrent s'ils ont des amis communs. Nous appelons cette constante `a` et lui donnons la valeur 0.5 par exemple.

```
DEFINE_CONST (a, 0.5);
```

- p_0 est la probabilité que deux individus se rencontrent s'ils n'ont aucun ami commun. Nous appelons cette constante `p0` et lui donnons la valeur de 0.006 par exemple.

```
DEFINE_CONST (p0, 0.006);
```

- k est un paramètre ajustable du modèle qui influence le calcul de la force de l'amitié entre deux contacts qui diminue au cours du temps si ceux-ci ne se rencontrent pas. Nous appelons cette constante `k` et l'initialisons à 0.01 par exemple.

```
DEFINE_CONST (k, 0.01);
```

- Un paramètre qui spécifie la limite minimum de la force de l'amitié entre deux contacts. Sous cette limite, le lien d'amitié n'existe plus. Nous appelons cette constante `limit_min_for_friendship` et l'initialisons à 0.5 par exemple.

```
DEFINE_CONST (limit_min_for_friendship, 0.5);
```

- Une approximation du nombre naturel e.

```
DEFINE_CONST (e, 2.718);
```

Définitions d'attributs

Ensuite, toujours en prenant l'exemple du modèle d'amitié, nous pourrions définir les deux attributs suivants sur l'entité link :

- S est la force de l'amitié entre deux contacts. Au début d'une nouvelle amitié (création d'un nouveau lien), la force est égale à 1 mais diminue au cours du temps si les deux amis ne se rencontrent pas. Nous appelons cet attribut `strength` et l'initialisons à 1.

```
DEFINE_ATTRIB (link, strength, 1);
```

- Δt est l'intervalle de temps entre deux rencontres successives de deux amis. Au début d'une amitié (à la création d'un lien), la valeur de cet attribut est égale à 0, mais augmente de 1 à chaque pas de temps jusqu'à ce que les deux amis se rencontrent de nouveau et que Δt soit remis à 0. Nous appelons cet attribut `time_since_last_meeting` et l'initialisons à 0.

```
DEFINE_ATTRIB (link, time_since_last_meeting, 0);
```

Définitions de fonctions

Finalement, le modèle d'amitié définit les fonctions suivantes :

- La fonction $g(m) = 1-(1-p_0)e^{-am}$ qui représente l'augmentation attendue de la chance que deux individus se rencontrent s'ils ont un ou plusieurs amis communs. Le nombre m_{ij} est le nombre d'amis communs de i et j où i et j représentent deux amis.

```
DEFINE_FUNCTION (link, g)
      1 - (1 - p0) * e ** (-a * number_of_common_contacts());
```

- La fonction $f(z_i) = \frac{1}{e^{\beta(z_i - z^*)} + 1}$ qui ajuste la probabilité pour un contact i de se faire de nouveaux amis. Nous aurons besoin de deux fonctions : Une fonction pour le premier nœud du lien i et une fonction pour le deuxième nœud du lien j.

```
DEFINE_FUNCTION (link, fi)
      1 /(e ** (b * (start_node().degree() - z_limite)) + 1);

DEFINE_FUNCTION (link, fj)
      1 /(e ** (b * (end_node().degree() - z_limite)) + 1);
```

- La fonction $p_{ij} = f(z_i)f(z_j)g(m_{ij})$ qui permet de calculer la probabilité d'ajouter un lien d'amitié entre les contacts i et j. Cette fonction utilise les trois fonctions définies ci-dessus.

```
DEFINE_FUNCTION (link, pij)
      g()*fi()*fj();
```

- La fonction $s_{ij} = e^{-k\Delta t}$ qui diminue la force du lien d'amitié à chaque pas de temps où deux amis ne se rencontrent pas.

```
DEFINE_FUNCTION (link, s)
      e ** (-k * time_since_last_meeting));
```

La prochaine section explique comment écrire les règles du modèle à simuler dans le langage de modélisation NetSim. Nous pourrons alors finaliser la traduction de ce modèle.

3.4 Syntaxe d'écriture des règles du modèle

3.4.1 Les actions

Les règles du modèle sont des actions à exécuter sur le réseau, à chaque pas de temps. Ces règles sont habituellement exécutées selon certaines probabilités et/ou conditions qui peuvent dépendre de l'état (l'ensemble des valeurs des attributs) des entités du réseau. Les règles du modèle sont donc spécifiées par le langage de modélisation via les actions.

Les actions du langage de modélisation sont en somme les actions de manipulation d'un graphe. Ils se résument à ajouter, modifier ou supprimer des liens ou des nœuds et à modifier les attributs du réseau ou du monde.

Chaque action concerne une entité spécifique de la simulation. Cette entité est spécifiée par le deuxième terme de la commande d'action. Par exemple, la commande `PICK_WORLD` concerne l'entité world tandis que la commande `ADD_NODES` concerne l'entité node. Cette constatation est importante car les paramètres des actions doivent respecter la visibilité des

entités les unes par rapport aux autres. Le tableau suivant présente la liste des neuf actions disponibles dans le langage NetSim.

Tableau 3.4.1.1 Les actions du langage NetSim

Actions	Définition
`PICK_WORLD (...)`	Cette action permet de choisir l'entité world dans le but d'en modifier les attributs.
`PICK_NETWORK (...)`	Cette action permet de choisir l'entité network dans le but d'en modifier les attributs.
`PICK_LINKS (...)`	Cette action permet de choisir des liens dans le but d'en modifier les attributs.
`ADD_LINKS (...)`	Cette action permet d'ajouter des liens.
`CUT_LINKS (...)`	Cette action permet de supprimer des liens.
`REWIRE_LINKS (...)`	Cette action permet de changer le deuxième nœud de certains liens par un autre nœud choisi au hasard.
`PICK_NODES (...)`	Cette action permet de choisir des noeuds dans le but d'en modifier les attributs.
`ADD_NODES (...)`	Cette action permet d'ajouter des nœuds.
`CUT_NODES (...)`	Cette action permet de supprimer des noeuds.

Chaque action peut contenir une séquence de paramètres, chacun étant séparé par une virgule. Chaque paramètre peut être soit une **expression arithmétique**, soit une **expression booléenne** ou soit une **affectation**.

- Une expression arithmétique est :
  - Une fonction numérique
  - Un attribut

- Une constante
- Une expression mathématique construite à l'aide d'opérateurs arithmétiques, des fonctions et/ou des constantes et/ou des attributs.

- Une expression booléenne est :
  - Une fonction booléenne
  - une expression booléenne construite à l'aide des opérateurs booléens et des fonctions et/ou des constantes et/ou des attributs.

- Une affectation est une expression qui modifie un attribut par une expression arithmétique à l'aide de l'opérateur d'affectation. Seuls les attributs peuvent être modifiés (peuvent se situer à gauche de l'opérateur d'affectation).

Les paramètres des actions sont donc soit des valeurs numériques, soit des conditions ou soit des affectations.

En ce qui concerne une valeur numérique :

- Si elle est entre 0 et 1 inclusivement, elle est considérée comme une probabilité.
- Si elle est plus petite que 0, elle est considérée comme étant égale à 0 et donc comme une probabilité.
- Si elle est plus grande que 1, elle est considérée comme un nombre entier (s'il y a des décimales, elles seront tronquées).

Il est important de respecter la visibilité des entités les unes par rapport aux autres (voir figure 3.2.1). Par exemple, pour une action concernant l'entité world, il est interdit d'utiliser, dans les expressions définissant les paramètres de l'action, les fonctions ou les attributs qui sont définies sur les entités network, link et node car l'entité world ne voit qu'elle-même. Par contre, une action qui concerne l'entité link peut contenir des paramètres définis à l'aide de

fonctions ou d'attributs définis sur l'entité link, l'entité network et l'entité world car l'entité link se voit elle-même, voit l'entité network et l'entité world.

Les paramètres des actions sont en quelque sorte une manière de choisir les entités sur lesquelles l'action sera effectuée. Ils peuvent être vus comme une série d'épreuves que les entités concernées par l'action doivent gagner afin que l'action puisse y être appliquée. Ces épreuves sont appliquées séquentiellement dans l'ordre où elles sont écrites et aussitôt qu'une épreuve est échouée, l'action n'est pas appliquée sur l'entité courante. Supposons les définitions suivantes et examinons quelques exemples :

```
DEFINE_CONST (p1, 0.3);
DEFINE_CONST (nb, 3);

DEFINE_ATTRIB (world, flag, 0);
DEFINE_ATTRIB (network, ok, 0);
DEFINE_ATTRIB (link, force, 1);

DEFINE_FUNCTION (network, prob1)
      number_of_links() / ((number_of_nodes() * (number_of_nodes() -
      1)) / 2);

DEFINE_FUNCTION (link, prob2)
      number_of_common_contacts () / (start_node().degree() +
      end_node().degree());

DEFINE_FUNCTION (node, cond1)
      degree() > mean_degree();
```

Les actions PICK

Ce type d'action sert à choisir des entités afin d'en modifier les attributs.

Exemple 1

Considérons les actions suivantes:

```
PICK_WORLD();
PICK_NETWORK ();
PICK_LINKS();
PICK_NODES();
PICK_WORLD (flag := 1);
```

```
PICK_NETWORK (p1, number_of_nodes() >= 100, ok := 1);
```

Les quatre premières actions n'exécuteront rien, car bien qu'elles choisissent respectivement l'entité world, l'entité network, les liens (links) et les nœuds (nodes) du réseau, elles n'ont aucun paramètre qui pourrait modifier les attributs de ces entités.

La cinquième action, par contre, aura pour effet de choisir l'entité world, sans condition, et de modifier sont attribut `flag` en lui assignant la valeur 1.

La dernière action choisira l'entité network si, tout d'abord, l'épreuve probabiliste indiquée par la constante `p1` est réussie et si ensuite, la condition booléenne qui stipule que le nombre de nœuds dans le réseau doit être plus grand ou égal à 100 est respectée. Si et seulement si ces deux tests sont réussis, l'attribut `ok` sera modifié par la valeur 1 sinon, l'action n'aura aucun effet.

Exemple 2

Considérons maintenant les actions suivantes :

```
PICK_LINKS (prob1(), force := 1, flag := flag + 1);

PICK_LINKS (20, prob2(), force := force + 1, color := 8,
            2, color := 5);

PICK_NODES (cond1(), fill_color := 6, nb, radius := 10);
```

La première action sera interprétée comme suit : tout d'abord, chacun des liens existants dans le réseau passera le test probabiliste donné par la fonction `prob1()`. Ensuite, seul l'attribut `force` des liens choisis prendra la valeur 1. Le dernier paramètre qui modifie l'attribut `flag`, défini sur l'entité world, pourrait agir ici comme un compteur des liens sélectionnés, car à chaque lien choisi, la valeur de `flag` augmente de 1. Il faudrait s'assurer, dans ce cas, que l'attribut `flag` soit bien initialisé à zéro en écrivant l'action `PICK_WORLD (flag := 0);` juste avant cette action `PICK_LINKS`.

La deuxième action commence tout d'abord par sélectionner 20 liens au hasard dans le réseau. Ensuite, chacun de ces 20 liens devra passer le test probabiliste donné par la fonction `prob2()`. Supposons que nous obtenons 12 liens gagnants. L'attribut `force` des ces 12 liens sera alors incrémenté de 1 et l'attribut `color` deviendra égal à 8. Finalement, parmi ces 12 liens, 2 liens seront choisis au hasard et la valeur de leur attribut `color` deviendra égale à 5.

La dernière action débute en sélectionnant, parmi tous les noeuds existants dans le réseau, les noeuds qui répondent à la condition donnée par la fonction cond1(). Ensuite, l'attribut fill_color de tous les nœuds choisis deviendra égale à 6. Enfin, parmi tous les nœuds déjà sélectionnés, on en choisira `nb` (= 3) au hasard qui verront la valeur de leur attribut `radius` être remplacée par la valeur 10.

Les actions ADD

Ces actions servent à ajouter des nœuds ou des liens dans le réseau selon certaines probabilités et/ou conditions.

Exemple 1

```
ADD_NODES();
ADD_LINKS();
```

Par convention, pour une action `ADD_NODES()` sans paramètres, on ajoute un nombre de noeuds égal au nombre de nœuds déjà présents dans le réseau au moment de l'exécution de l'action. Si cette action, sans paramètre, est exécutée dans un réseau vide, elle n'a donc aucun effet. C'est pourquoi, lors de l'initialisation d'un réseau, nous utilisons souvent cette action avec un nombre comme paramètre. Par exemple, `ADD_NODES (100)` ajoutera 100 nœuds au réseau. Considérons maintenant les deux actions suivantes:

```
ADD_NODES (100);
ADD_NODES ();
```

Après l'exécution de ces deux actions, il y aura 200 nœuds dans le réseau.

L'action `ADD_LINKS()`, sans paramètre, ajoutera tous les liens potentiels possibles dans le réseau c'est-à-dire, tous ceux qui peuvent être ajoutés dans le réseau jusqu'à ce que le graphe soit complet.

Exemple 2

```
ADD_NODES (300, number_of_nodes() < 500);
ADD_NODES (p1, 100);
ADD_NODES (100, p1, border_color := 4);
```

La première action crée d'abord 300 nouveaux nœuds et, selon le deuxième paramètre, les ajoutera au réseau tant que le nombre total de nœuds dans le réseau est plus petit que 500.

La deuxième action a comme premier paramètre une probabilité. Premièrement, elle crée un nombre de nouveaux nœuds égal au nombre de nœuds déjà présents dans le réseau et chacun de ces nouveaux nœuds doit ensuite passer le test probabiliste donné par `p1`. Parmi les nœuds gagnants, elle en sélectionne aléatoirement 100 qui sont finalement ajoutés au réseau.

La dernière action ajoute d'abord 100 nouveaux nœuds au réseau. Ensuite, l'attribut `border_color` des nœuds ayant passé le test probabiliste donné par `p1` prend la valeur 4 et ces nouveaux nœuds sont finalement ajoutés au réseau.

Exemple 3

```
ADD_LINKS (200);

ADD_LINKS (200, ok := 1);

ADD_LINKS (prob2(), force := force + 1, 10,
           end_node().cond1(), force := force + 1);
```

La première action crée 200 liens potentiels et les ajoute au réseau.

La deuxième action crée d'abord tous les liens potentiels du réseau. Ensuite, chaque lien potentiel qui passe l'épreuve probabiliste donnée par la fonction `prob2()` verra son attribut `force` augmenter de 1. Puis, parmi les liens précédemment sélectionnés, 10 liens sont

choisis. Ces 10 liens sont ensuite testés sur leur nœud d'arrivée (`end_node()`) selon la condition `cond1()` et les liens qui répondent à cette condition verront leur `force` augmenter encore de 1.

Les actions CUT

Les actions de ce type servent à supprimer des nœuds ou des liens du réseau selon certaines probabilités et/ ou conditions. Les actions `CUT_NODES()` et `CUT_LINKS()` sans paramètre suppriment respectivement tous les nœuds et les liens du réseau. Lorsque ces actions ont des paramètres, elles agissent comme les actions `PICK_NODES()` et `PICK_LINKS()` à l'exception que les nœuds ou les liens gagnants de toutes les épreuves données par les paramètres de l'action sont alors supprimés du réseau.

L'action `REWIRE_LINKS`

Cette action consiste à changer le nœud d'arrivée d'un lien (`end_node()`) par un autre nœud du réseau choisi au hasard. L'action `REWIRE_LINKS()`, sans paramètre, a pour effet de changer le nœud d'arrivée de tous les liens du réseau. Sinon, lorsqu'elle a des paramètres, cette action fonctionne de la même manière que l'action `CUT_LINKS(...)` à l'exception que tous les liens ayant gagné toutes les épreuves données par les paramètres de l'action sont reconnectés au lieu d'être supprimés.

3.4.2 Commandes d'exécution de la simulation

Arrêt de la simulation

Le langage de modélisation NetSim offre aussi une instruction qui permet de spécifier le moment d'arrêt de la simulation. Le nom de cette commande est `SIMUL_STOP_CONDITION` et elle prend une expression booléenne comme seul paramètre. Cette expression peut donc être formée d'opérateurs booléens et/ou de constantes et/ou d'attributs et/ou de fonctions pourvu que ceux-ci respectent les règles de visibilité. À cet effet, nous devons considérer la

commande d'arrêt comme si elle était définie sur l'entité network. Celle-ci peut donc contenir, dans l'expression de son paramètre, des attributs et/ou des fonctions définis sur les entités network et world.

La simulation se termine aussitôt que l'expression booléenne devient vraie. Dit d'une autre façon, la simulation s'exécutera tant que la condition n'est pas vraie. Si aucune commande d'arrêt de la simulation n'est donnée, la simulation ne s'arrêtera pas automatiquement. Certaines fonctionnalités du logiciel NetSim permettent cependant de terminer la simulation manuellement, via l'interface graphique.

Par exemple, pour que la simulation s'arrête au pas de temps 1000, nous pourrions écrire :

```
SIMUL_STOP_CONDITION (time_step() > 1000);
```

Pour que la simulation se termine aussitôt que le degré moyen de tous les nœuds du réseau dépasse la valeur 5, par exemple, nous pourrions écrire :

```
SIMUL_STOP_CONDITION (mean_degree() > 5);
```

Phases d'exécution

Une phase, dans le langage NetSim, est un groupe de règles (d'actions) qui sont exécutées ensemble à chaque pas de temps. Il est possible de définir plusieurs phases dans une simulation. Par exemple, nous pourrions tout d'abord construire un réseau initial avec un ensemble particulier de règles et, à un moment spécifié, continuer la simulation avec un groupe de règles différentes. Nous pourrions aussi définir différentes phases et les exécuter selon certaines probabilités à chaque pas de temps.

Il faut donc spécifier une condition qui détermine si une phase s'exécutera ou non, à un pas de temps donné. L'instruction `PHASE_EXEC_CONDITION` prend une expression booléenne comme seul paramètre. La phase ne s'exécutera que si le résultat de l'évaluation de son expression booléenne est vrai.

Il est à noter que seule la première phase ne nécessite pas la spécification d'une condition d'exécution. Si la commande `PHASE_EXEC_CONDITION` est absente, pour la première phase, la phase sera exécutée sans condition. Pour la définition de toutes les phases subséquentes, la condition d'exécution est requise, car cette commande détermine le début d'une nouvelle phase. La figure 3.4.2.1 montre un fichier d'actions contenant trois phases d'exécution différentes et une commande d'arrêt automatique de la simulation. Selon cet exemple, la simulation s'arrêtera au pas de temps 100. La première phase regroupe trois actions et s'exécute sans condition. La deuxième phase contient deux actions et s'exécute si le pas de temps est plus grand que zéro et finalement, la troisième phase, qui définit deux actions, s'exécute selon une probabilité de 0.7.

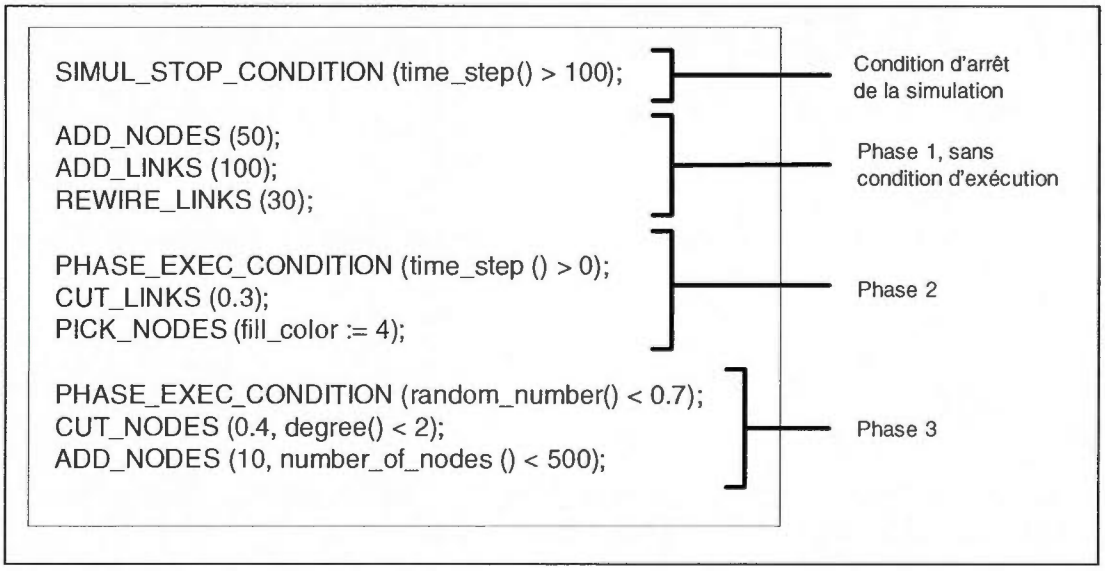


Figure 3.4.2.1 Phases d'exécution et commande d'arrêt automatique de la simulation

Ceci nous amène à parler des deux modes d'exécution de phases que gère NetSim : le mode séquentiel et le mode parallèle.

Selon le mode séquentiel, les phases s'exécutent, une après l'autre, dans l'ordre où elles sont écrites. Pour qu'une phase puisse être exécutée, il faut absolument que la phase précédente ait d'abord été exécutée. Au premier pas de temps, la simulation tente d'exécuter la première phase. Si celle-ci n'a pas de condition d'exécution, elle s'exécute jusqu'à ce que la condition d'exécution de la seconde phase soit vraie. Ensuite, la seconde phase s'exécute jusqu'à ce

que la condition d'exécution de la troisième phase soit vraie et ainsi de suite. Si la première phase définit une condition d'exécution, le programme évalue la condition et exécute la première phase seulement si le résultat de l'évaluation est vrai. Si la condition n'est jamais vraie, la simulation ne fait rien, car elle reste bloquée sur cette phase.

Le pseudo-code suivant décrit l'algorithme de la simulation des trois phases décrites à la figure 3.4.2.1 en mode séquentiel.

```
CompteurDePhases = 0
pasDeTemps = 0

TANT QUE pasDeTemps <= 100 FAIRE

      # condition d'exécution de la phase 3
      SI nombreAléatoire < 0.7 et compteurDePhases = 2 ALORS
            compteurDePhases = 3
            exécuter phase 3

      # condition d'exécution de la phase 2
      SINON SI le pasDeTemps > 0 et compteurDePhases = 1 ALORS
            compteurDePhases = 2
            exécuter phase 2

      # exécution de la phase 1
      SINON SI compteurDePhases = 0 ALORS
            compteurDePhases = 1
            exécuter phase 1
      FIN SI

      pasDeTemps = pasDeTemps + 1

FIN TANT QUE
```

Le mode d'exécution en parallèle fonctionne de façon tout à fait différente. À chaque pas de temps, chaque phase s'effectue si le résultat de l'évaluation de sa condition d'exécution est vrai. Le pseudo-code suivant décrit l'algorithme de la simulation des trois phases décrites à la figure 3.4.2.1 en mode parallèle.

```
pasDeTemps = 0

TANT QUE pasDeTemps <= 100 FAIRE

      Exécuter phase 1

      SI pasDeTemps > 0 ALORS
            Exécuter phase 2
      FIN SI

      SI nombreAléatoire() < 0.7 ALORS
            Exécuter phase 3
      FIN SI

      pasDeTemps = pasDeTemps + 1

FIN TANT QUE
```

Le choix du mode d'exécution des phases dépend du genre de modèle que l'on veut simuler. La spécification du mode se fait via l'interface graphique du logiciel NetSim au moment de la création d'une simulation.

3.4.3 Exemple d'écriture des règles du modèle d'amitié de Jin, Girvan et Newman

Nous allons reprendre l'exemple du modèle d'amitié et allons maintenant écrire les règles de ce modèle en langage de modélisation NetSim.

Une des règles du modèle considère un nombre fixe de noeuds. Cette règle indique simplement qu'il n'y aura pas d'ajout ni de suppression de nœuds au cours de la simulation et donc, pas d'actions `ADD_NODES` ou `CUT_NODES`.

Une autre règle concerne la formation de nouveaux liens d'amitié. D'un côté, le nombre d'amis qu'un individu peut posséder doit diminuer rapidement lorsque son nombre de relations atteint la constante z^* (degré limité). Ce facteur est représenté par la fonction $f(Z_i)$. De l'autre côté, la probabilité que deux individus créent un lien d'amitié doit être plus élevée s'ils ont un ou plusieurs amis communs (transitivité). Ce facteur est représenté par la fonction $g(m)$. L'ajout de liens, dans le réseau, doit donc se faire selon une certaine

probabilité qui tient compte de ces deux facteurs et qui est représentée par la fonction p_{ij} que nous avons déjà définie. Nous pourrions donc définir cette règle comme suit :

```
ADD_LINKS(pij());
```

Selon cette action, chaque lien potentiel du réseau doit passer le test probabiliste donné par la fonction `pij()`. Chaque lien qui passe l'épreuve sera ajouté au réseau.

Les dernières règles du modèle d'amitié concernent la suppression de liens d'amitié qui ne sont pas entretenus au cour du temps. (1) À chaque pas de temps, deux amis qui ne se rencontrent pas voient la force de leur amitié diminuer exponentiellement selon $s_{ij} = e^{-k\,\Delta t}$. Nous avons déjà écrit cette fonction `s()` dans les définitions. (2) À chaque pas de temps, deux amis qui se rencontrent de nouveau renforcent leur amitié (force := 1). (3) Lorsque la force d'une amitié devient trop petite (plus petite que la constante `limit_min_for_friendship`), le lien d'amitié disparaît.

Nous pourrions écrire ces règles comme suit :

```
(1) PICK_LINKS(time_since_last_meeting := time_since_last_meeting +
               1, strength := s());

(2) PICK_LINKS(pij(), time_since_last_meeting := 0 ,
               strength := 1);

(3) CUT_LINKS (strength <= limit_min_for_friendship);
```

L'action (1) sélectionne tous les liens du réseau pour ajuster les attributs `time_since_last_meeting` et `strength` de chaque lien. En effet, à chaque pas de temps, la force de la relation entre chaque couple d'amis diminue si ceux-ci ne se rencontrent pas de nouveau. L'attribut `time_since_last_meeting` comptabilise, pour chaque relation, le nombre de pas de temps qui se sont écoulés depuis la dernière rencontre de deux amis.

L'action (2) modélise la rencontre de certains couples d'amis qui se connaissent déjà et renforcent ainsi leur amitié. Elle choisit, parmi tous les liens du réseau, ceux qui passent le test probabiliste donné par la fonction `pij()`. Ensuite, pour chaque lien sélectionné, elle remet l'attribut `time_since_last_meeting` à 0 et l'attribut `strength` à 1 pour indiquer que cette relation a été renouvelée.

L'action (3) sert à décrire la règle qui stipule qu'une relation d'amitié tend à disparaître si elle n'est pas entretenue. Dans le modèle, la constante `limit_min_for_friendship` représente la limite inférieure permise sur la force d'une relation avant que celle-ci ne cesse d'exister. Cette action supprime donc tous les liens d'amitié dont la force (`strength`) est plus petite ou égale à `limit_min_for_friendship`.

3.5 Conclusion

Le langage de modélisation NetSim, bien qu'il demande un certain apprentissage, est un langage très simple en comparaison au langage de programmation usuel. Dans cette optique, nous croyons qu'il est assez abordable pour des utilisateurs variés.

De plus, comme on a pu le constater en traduisant le modèle du réseau d'amis, la description d'un modèle en langage NetSim ne demande pas beaucoup de lignes de commandes et ces commandes se modifient très facilement lorsque l'on désire expérimenter divers paramètres ou différentes règles lors de nos simulations. Ceci n'est pas à négliger sachant qu'il faut souvent plusieurs essais avant de trouver les bonnes règles et les bonnes valeurs de paramètres lors de la construction de modèles dynamiques.

Bien que le langage de modélisation NetSim ait été conçu plus particulièrement pour la modélisation de réseaux sociaux et du Web, nous croyons en son potentiel de développement quant à la modélisation d'une grande variété de réseaux d'information.

CHAPITRE 4

NetSim : une plate-forme de modélisation et simulation de Réseaux d'information

4.1 Introduction

NetSim est une plate-forme de simulation capable d'interpréter le langage de modélisation NetSim. Étant donné la flexibilité de ce langage, il est possible de simuler facilement divers types de modèles de réseaux d'information. Dans ce chapitre, nous présentons tout d'abord une vue générale du fonctionnement de NetSim pour ensuite expliquer certains détails d'implémentation que nous considérons comme les plus importants.

4.2 Fonctionnement général de NetSim

NetSim est une application qui sert à interpréter le langage de modélisation NetSim. Cependant, ce logiciel n'est pas un interpréteur en soi. Il est basé sur le langage Java et se sert de la machine virtuelle Java pour interpréter le langage de modélisation, préalablement traduit en code Java. La figure 4.2.1 illustre les grandes étapes du fonctionnement de NetSim.

La première étape de la construction d'une simulation dans NetSim consiste à écrire les définitions et les règles du modèle à simuler en langage de modélisation NetSim, comme nous l'avons fait au chapitre 3. Deuxièmement, l'application vérifie la syntaxe de ces règles et définitions et, lorsqu'il n'y a pas d'erreur, le programme lit le modèle NetSim en mémoire. Ensuite, l'application traduit le modèle NetSim en modèle Java et d'après ce modèle Java, elle écrit en mémoire les classes d'exécution de la simulation. Finalement, ces classes sont

compilées en instructions (bytecodes) compréhensibles pour la machine virtuelle Java de sorte que celle-ci puisse interpréter la simulation.

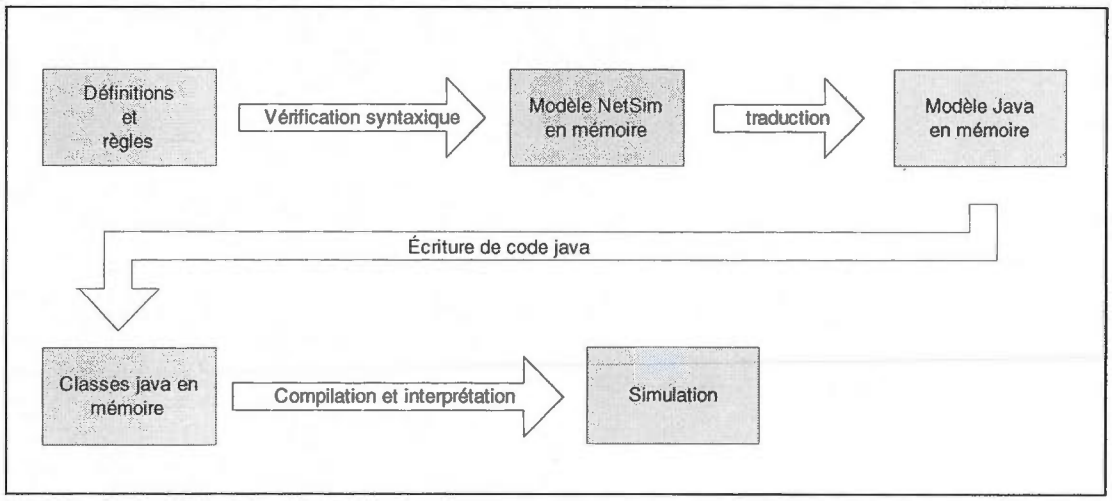


Figure 4.2.1 Fonctionnement général de NetSim

La figure 4.2.2 illustre un diagramme de classes UML qui montre les classes et associations principales nécessaires à la construction d'une simulation. L'objet `ApplicationController` est le premier à être créé. C'est en quelque sorte celui qui orchestre la mise en œuvre de la simulation.

Premièrement, l'`ApplicationController` lit les définitions et les règles du modèle écrit en langage de modélisation NetSim. Ensuite, la méthode `verifyDefinitionsSyntax()` de l'objet `ApplicationController` construit le `SyntaxVerificator` de l'objet `SimulationData` en lui passant en paramètre les définitions et les règles du modèle. Elle appelle ensuite la méthode `verifyDefs()` sur l'objet `SyntaxVerificator` qui vérifie si la syntaxe des définitions du modèle est valide. Si elle est valide, le `SyntaxVerificator` crée l'objet `ModelDefinitions` et y entrepose les définitions NetSim valides. Si la syntaxe n'est pas respectée, l'attribut `errorMsgDefs` du `SyntaxeVerificator` contiendra un message d'erreur significatif généré par la méthode `verifyDefs()`.

Ensuite, si les définitions NetSim sont syntaxiquement correctes, l'`ApplicationController` appelle sa méthode `verifyRulesSyntax()` qui appelle à son tour la méthode `verifyRules()` du `SyntaxVerificator`. La syntaxe des règles NetSim est alors vérifiée en considérant les définitions préalablement entreposées dans le `ModelDefinitions`. S'il y a erreur de syntaxe, l'attribut `errorMsgRules` du `SyntaxVerificator` contiendra un message d'erreur significatif généré par la méthode `verifyRules()`.

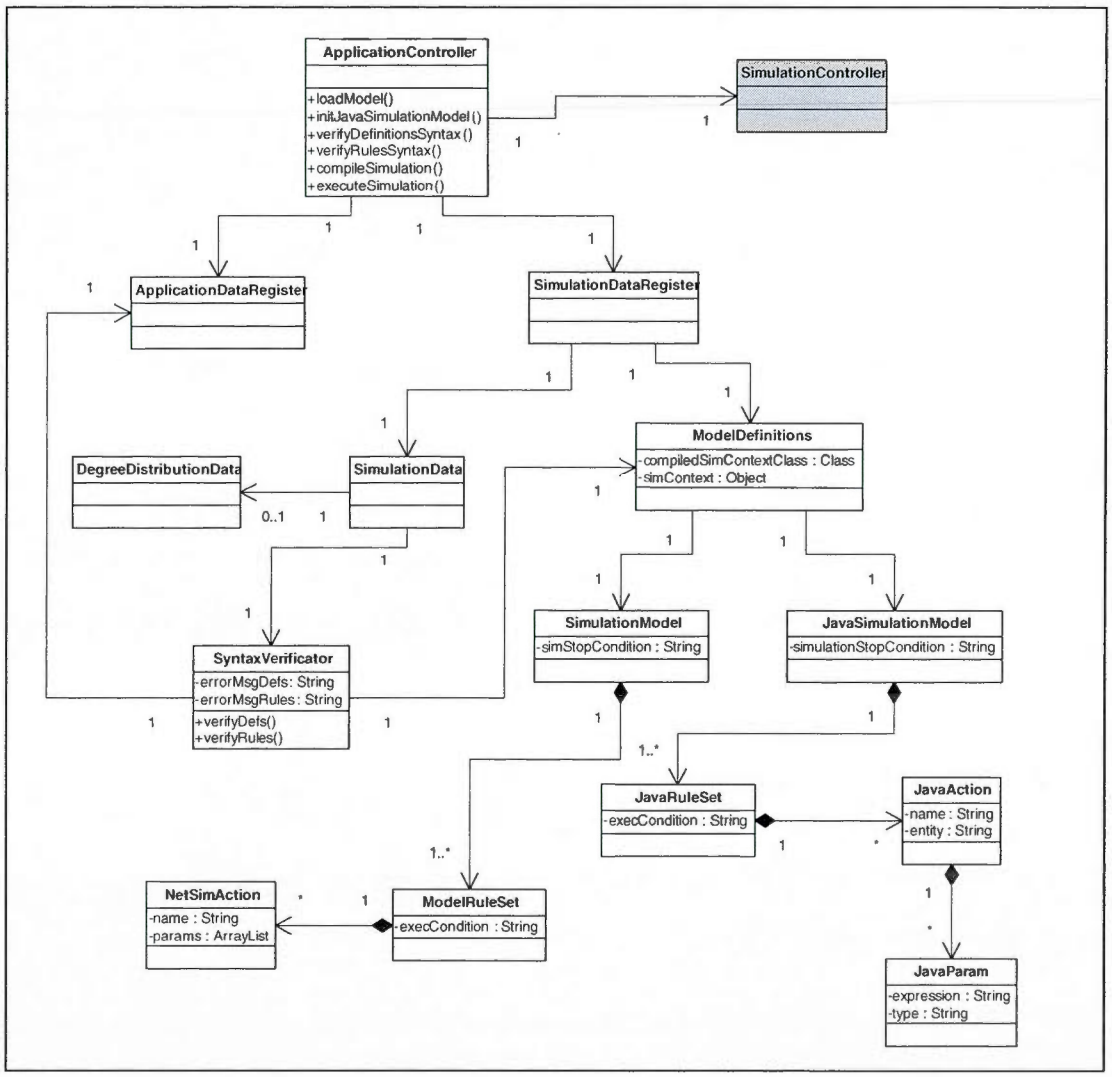


Figure 4.2.2 Diagramme de classes UML des classes principales nécessaires à la construction d'un modèle dans NetSim

Les définitions NetSim sont entreposées dans le `ModelDefinitions` sous forme de dictionnaires (HashMap). Par exemple, les définitions du modèle d'amitié, que nous avons écrites en langage NetSim au chapitre précédent, ressembleraient à ceci une fois en mémoire :

Définitions des constantes :

```
z_limite                  ---> 5.0
k                         ---> 0.01
a                         ---> 0.5
p0                        ---> 0.006
limit_min_for_friendship  ---> 0.5
b                         ---> 5.0
e                         ---> 2.718
```

Définitions des attributs sur l'entité link :

```
strength                ---> 1.0
time_since_last_meeting ---> 0.0
```

Définitions des fonctions sur l'entité link :

```
pij ---> g()*fi()*fj()

fi ---> 1 /(e ** (b * (start_node().degree() - z_limite)) + 1)

fj ---> 1 /(e ** (b * (end_node().degree() - z_limite)) + 1)

g ---> 1 - (1 - p0) * e ** (-a * number_of_common_contacts())

s ---> e ** (-k * time_since_last_meeting)
```

Si la syntaxe des règles est valide, avec sa méthode `loadModel()`, l'`ApplicationController` lit en mémoire toutes les règles (actions) écrites en langage NetSim et construit le modèle NetSim. Ce modèle est composé de trois classes : `SimulationModel`, `ModelRuleSet` et `NetSimAction`. La classe `SimulationModel` contient un attribut de type `String` qui contient l'expression, en langage NetSim, de la condition d'arrêt de la simulation (qui peut être nulle). Elle contient

aussi une liste de toutes les phases du modèle où chacune comprend une ou plusieurs actions NetSim et une condition d'exécution (qui peut être nulle pour la première phase). C'est l'objet `ModeRuleSet` qui sert à stocker les informations d'une phase. Ce dernier possède comme attribut l'expression, en langage NetSim, de la condition d'exécution de la phase et une liste de toutes les actions NetSim contenues dans cette phase. Chaque action NetSim est ensuite décomposée en deux objets : une chaîne de caractère qui est le nom de l'action et une liste de chaînes de caractères qui contient tous les paramètres de l'action écrits en langage NetSim.

Pour donner un exemple plus concret de la création du `SimulationModel`, nous allons nous inspirer du modèle de réseau d'amis écrit au chapitre précédent. Supposons d'abord que nous voulons que la simulation s'arrête au temps t = 1000. Ensuite, nous voulons, dans un premier temps, construire un réseau initial de 250 nœuds, de façon aléatoire, en ajoutant des liens jusqu'à ce que le degré moyen du réseau soit égal à `z_limite`. Pour ce faire, nous écrivons une première phase, sans spécifier de condition d'exécution, qui ajoute 250 nœuds au temps t = 0. La deuxième phase a comme condition d'exécution que le pas de temps soit supérieur à 0 et entre donc en action au temps t = 1. Cette deuxième phase ne contient qu'une seule action qui ajoute 10 liens, au hasard, à chaque pas de temps. La troisième phase est déclenchée au moment où le degré moyen du réseau est plus grand que `z_limite` et contient les règles (actions) du modèle telles qu'écrites au chapitre précédent. Voici les actions NetSim qui modélisent ce qui vient d'être discuté (nous supposons un mode d'exécution séquentiel) :

```
#condition d'arrêt de la simulation
SIMUL_STOP_CONDITION (time_step () >= 1000);

#première phase, sans condition d'exécution
ADD_NODES (250);


#deuxième phase
PHASE_EXEC_CONDITION (time_step() > 0);

ADD_LINKS (10);'

#troisième phase
PHASE_EXEC_CONDITION (mean_degree() > z_limite);
```

```
PICK_LINKS(time_since_last_meeting := time_since_last_meeting + 1,
           strength := s());

PICK_LINKS(pij(), time_since_last_meeting := 0 ,
                  strength := 1);

CUT_LINKS (strength <= limit_min_for_friendship);

ADD_LINKS(pij());
```

La figure 4.2.3 illustre les objets en mémoire qui contiennent les informations de ce modèle écrit en langage de modélisation NetSim.

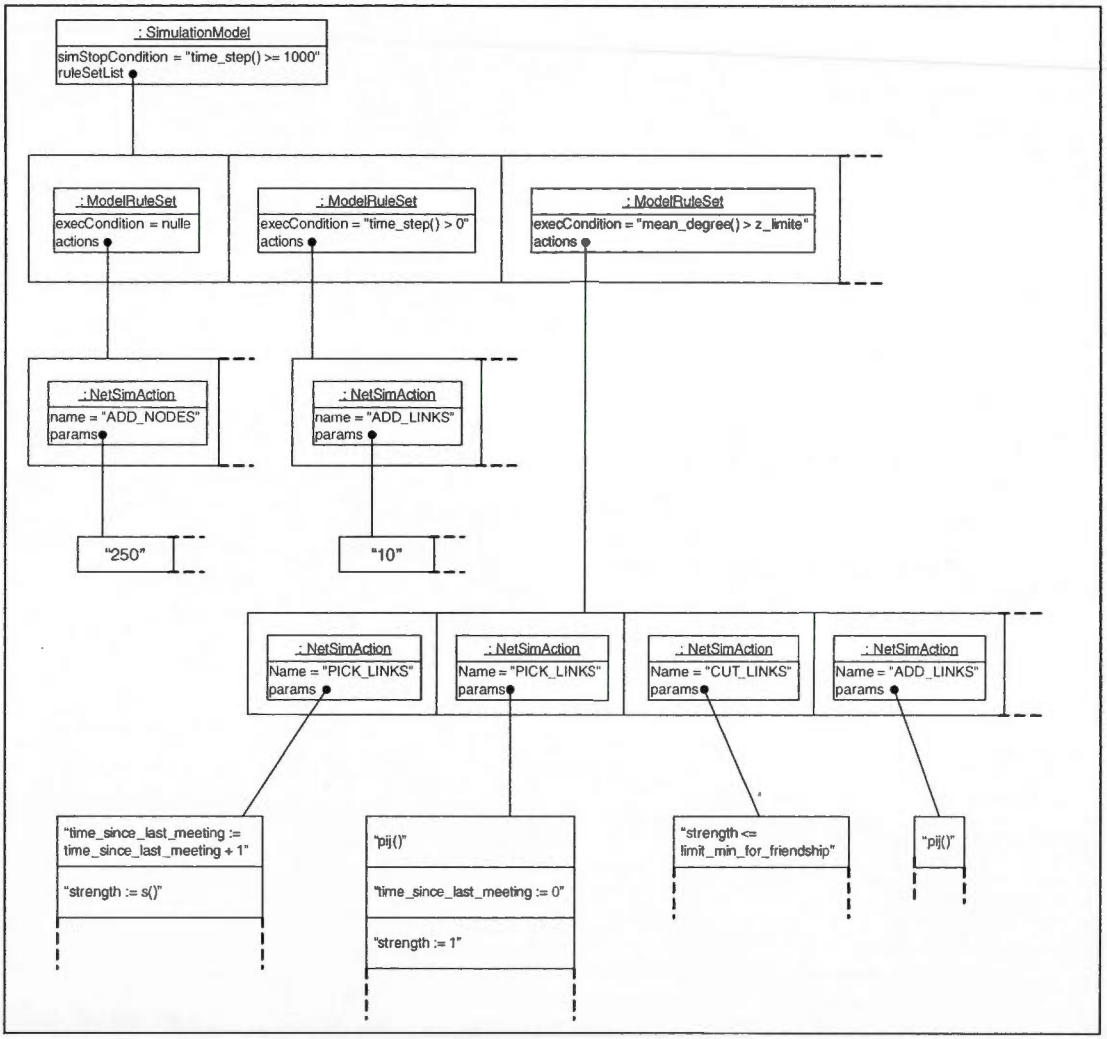


Figure 4.2.3 Représentation en mémoire d'un modèle NetSim du réseau d'amis

Après avoir créé le modèle NetSim en mémoire, l'`ApplicationController` appelle sa méthode `initJavaSimulationModel()` qui traduit toutes les expressions du modèle NetSim en expression Java dans un modèle composé de quatre classes : `JavaSimulationModel`, `JavaRuleSet`, `JavaAction` et `JavaParam`. Le `JavaSimulationModel` est similaire au `SimulationModel` à l'exception que toutes les expressions NetSim sont traduites en expressions Java.

Toutes les fonctions prédéfinies du langage de modélisation NetSim sont traduites par des méthodes Java définies dans la classe `SimulationController`. Prenons, par exemple, la première action de la troisième phase du modèle NetSim :

```
PHASE # 3
  condition exécution: "mean_degree() >  z_limite"

  ACTION #1
    Name : "pick_links"
    Params :
      "time_since_last_meeting := time_since_last_meeting + 1"
      "strength := s()"
```

Supposons que l'objet `simController` est une instance de la classe `SimulationController`.

Traduction de la condition d'exécution :

La méthode NetSim « `mean_degree()` » est traduite par un appel de la méthode `getMeanDegree()` sur l'objet simController.

```
simController.getMeanDegree() > 5.0
```

Traduction du premier paramètre de l'action # 1

L'opérateur d'affectation NetSim « := » d'un attribut défini sur l'entité link est traduit par la méthode :

```
setAttribute(Link link, String nomAttrib, double valeurAttrib).
```

Pour obtenir la valeur d'un attribut défini sur l'entité link, il faut appeler, sur l'objet `simController`, la méthode :

```
getAttribute (Link link, String nomAttrib)
```

Ainsi, la traduction complète est :

```
simController.setAttribute(link, "time_since_last_meeting",
      simController.getAttribute(link, "time_since_last_meeting") +
      1.0)
```

Traduction du deuxième paramètre de l'action # 1

On remarque que la fonction `s()` a été remplacée par son expression correspondante avant d'être traduite en code Java.

```
simController.setAttribute(link, "strength",
      ( Math.pow(2.718, - 0.01 * simController.getAttribute(link,
      "time_since_last_meeting")) ))
```

Nous ne détaillerons pas ici la procédure complète de traduction des expressions NetSim en expressions Java, mais nous en reparlerons à la section 4.3.2.

Une fois le modèle NetSim traduit en Java, selon que le mode d'exécution de la simulation choisi est séquentiel ou parallèle, NetSim écrit le code Java des classes nécessaires à l'exécution de la simulation. Nous verrons cette partie plus en détail à la section 4.3.3.

Le `SimulationController` est l'objet qui contrôle la simulation. Il contient toutes les méthodes qui sont appelées par les classes responsables de l'exécution de la simulation : les méthodes d'ajout et de suppression de nœuds ou de liens du réseau, les méthodes de consultation et de modification des attributs de toutes les entités du modèle simulé ainsi que des méthodes d'arrêt, de mise en marche, d'arrêt temporaire, et de reprise de la simulation. De plus, c'est lui qui est responsable de la coordination de l'état du graphe représentant le

réseau simulé et la visualisation de l'évolution de ce réseau à l'écran. La figure 4.2.4 montre les classes principales qui contrôlent et implémentent la simulation.

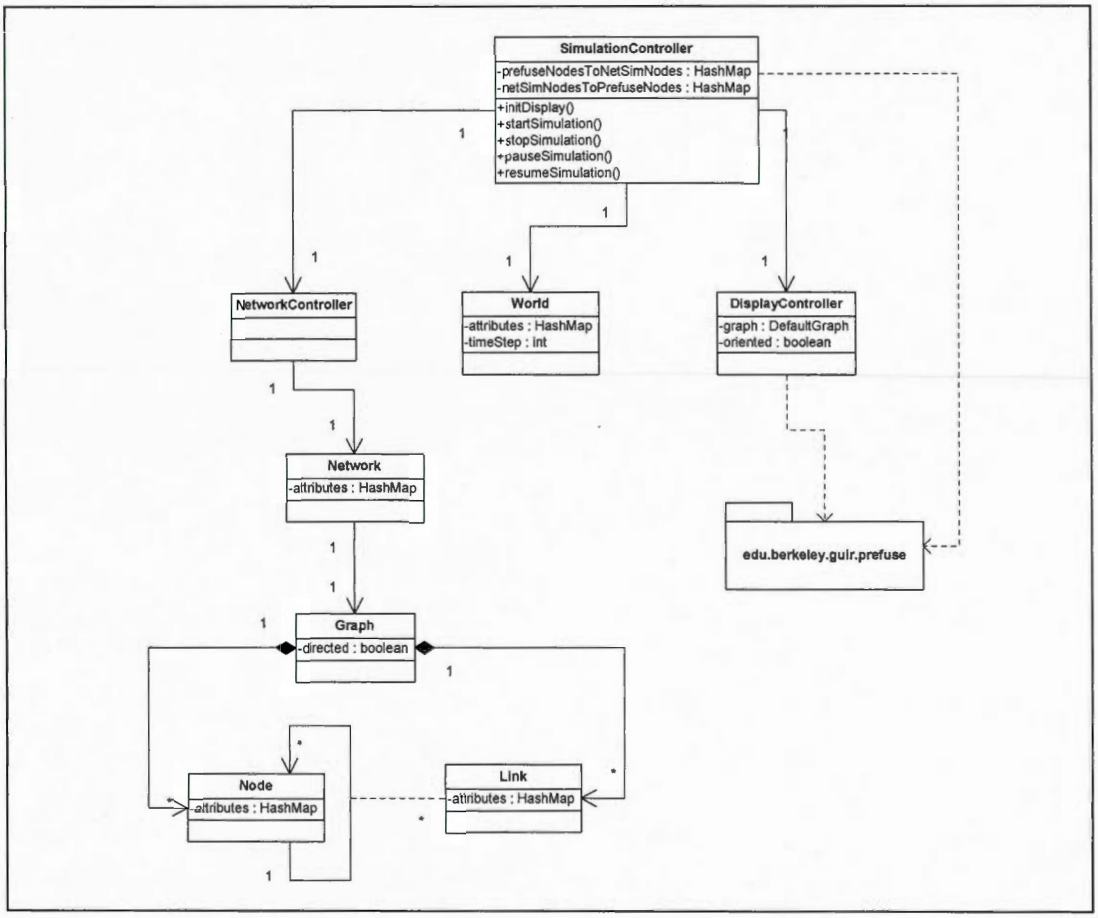


Figure 4.2.4 Diagramme de classes UML des principales classes de contrôle de la simulation

Dans la classe `SimulationController`, la méthode `initDisplay()` est responsable de l'initialisation de la fenêtre de visualisation du réseau, la méthode `startSimulation()` démarre la simulation et la méthode `stopSimulation()` arrête la simulation.

La classe `SimulationController` contient deux attributs très importants. Le premier est une instance de la classe `NetworkController` qui contient et gère le graphe représentant le réseau qui évolue dans le temps, selon les règles du modèle. Le deuxième

attribut est une instance de la classe `DisplayController` qui est responsable de l'affichage du réseau à l'écran.

Pour ce qui est de la visualisation du réseau, nous avons utilisé la librairie Prefuse (http://www.prefuse.org/). Cette librairie offre des classes servant à la manipulation d'un graphe et à sa visualisation à l'écran. Il faut construire le graphe avec les classes `DefaultGraph`, `DefaultNode` et `DefaultLink` fournies dans le paquetage edu.berkeley.guir.prefuse, afin qu'il puisse être visualisé à l'écran. Comme nous ne voulions pas que notre application dépende du graphe d'une librairie particulière, au cas où nous voudrions éventuellement utiliser une autre librairie, nous avons construit notre propre graphe. La figure 4.2.5 illustre la structure de données que nous avons utilisée pour construire notre graphe.

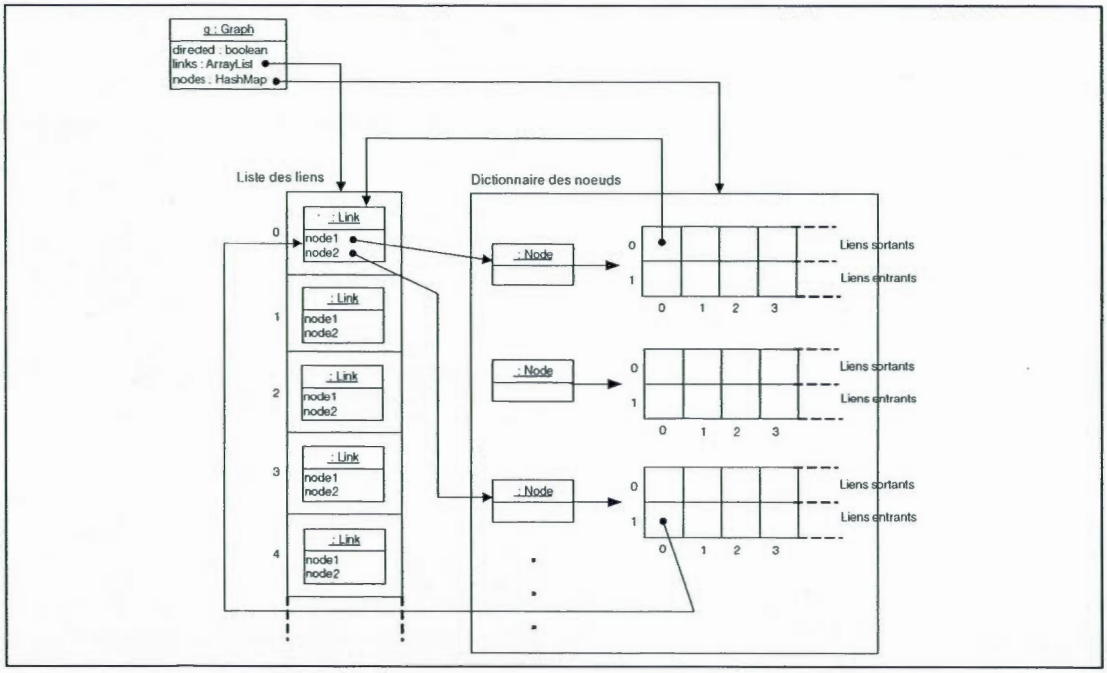


Figure 4.2.5 Structure de données représentant le graphe de NetSim en mémoire
Cette figure montre la liste de liens et le dictionnaire des nœuds du graphe. On remarque qu'un même lien (pour un graphe orienté) sera présent dans la liste des liens sortants d'un nœud ainsi que dans la liste des liens entrants d'un autre nœud puisqu'un lien est formé d'un nœud de départ et d'un nœud d'arrivée.

Le graphe de NetSim est composé d'une liste de nœuds et d'une liste de liens. La liste de noeuds est plus particulièrement un dictionnaire (java.util.HashMap) dans lequel les clés sont les nœuds et les valeurs correspondantes sont des tableaux de longueur deux. La première case du tableau contient la liste des liens sortants du nœud clé. La deuxième case du tableau contient la liste des liens entrants sur le nœud clé. Si le graphe n'est pas dirigé, seule la première case du tableau est utilisée et contient une liste des liens (sortants ou entrants) du nœud clé.

C'est le `SimulationController` qui gère la correspondance entre les nœuds du graphe de NetSim et les nœuds du graphe de Prefuse et entre les liens du graphe de NetSim et ceux de Prefuse. Pour ce faire, il gère deux attributs de type dictionnaire (Java.util.HashMap) dans lesquels il conserve et maintient la correspondance des nœuds et des liens. Par exemple, en cours de simulation, pour ajouter un nœud au graphe, on appelle la méthode `addNode(Node n)` du `SimulationController`. Celui-ci se charge d'ajouter le nœud dans le graphe NetSim, via le `NetworkController` et puis il crée le nœud de Prefuse (`DefaultNode`) et se charge de l'ajouter dans le graphe à visualiser via le `DisplayController`. Ensuite, il ajoute la correspondance entre le nœud de NetSim et le nœud de Prefuse dans le dictionnaire des nœuds (nœud NetSim ---> nœud Prefuse). De la même manière, lors de la suppression d'un nœud, on appelle la méthode `cutLink(Node n)` du `SimulationController` qui trouve, dans le dictionnaire des nœuds, le nœud de Prefuse correspondant au nœud à retirer du graphe. La méthode retire ensuite le nœud du graphe de NetSim via le `NetworkController` et supprime le nœud du graphe de Prefuse via le `DisplayController`. Finalement, le dictionnaire des nœuds est mis à jour. Le même principe s'applique pour les liens.

Lorsque le `simulationController` ajoute un nœud ou un lien au réseau, il va tout d'abord chercher les attributs de ce nœud ou de ce lien qui sont stockés dans le `ModelDefinitions` sous forme de dictionnaire et les copie ensuite dans l'attribut `attributes` du nœud ou du lien à ajouter. Pour consulter la valeur d'un attribut, au cours de la simulation, c'est la méthode `getAttribute (Node n, String nomAttrib)` ou `getAttribute(Link l, String nomAttrib)` qui est appelée sur le

`SimulationController`. Ces méthodes retournent la valeur de l'attribut `nomAttrib` du nœud `n` ou du lien `l`. Pour modifier la valeur d'un attribut, c'est la méthode `setAttribute(Node n, String nomAttrib, double nouvelleValeur)` ou `setAttribute(Link l, String nomAttrib, double nouvelleValeur)` qui est appelée. Ces méthodes affectent `nouvelleValeur` à l'attribut `nomAttrib` du nœud `n` ou du lien `l`.

Les entités world et network du langage de modélisation NetSim sont représentées par les classes `World` et la classe `Network`. Chacune de ces deux classes contient aussi un dictionnaire (nomAttrib ---> valeur) pour gérer les valeurs de leurs attributs respectifs. De la même façon que pour les nœuds et les liens, la consultation et la modification de ces attributs se font via les méthodes `getAttribute(...)` et `setAttribute(...)` du `SimulationController`.

Nous allons maintenant voir certains aspects de l'implémentation de NetSim plus en détail.

4.3 Détail d'implémentation de NetSim

4.3.1 Initialisation du logiciel

NetSim possède plusieurs fichiers de configuration qui sont lus en mémoire lors de l'initialisation du logiciel. À l'exception du fichier XML de configurations initiales du réseau offertes par NetSim, les données de ces fichiers sont représentées en mémoire sous la forme d'un dictionnaire (HashMap) dans l'objet `ApplicationData`. Nous présentons ici les fichiers principaux.

Configuration initiale du réseau de la simulation

Avant de commencer une simulation, NetSim offre la possibilité à l'utilisateur de choisir une configuration initiale du réseau dans le cas où celui-ci ne voudrait pas que la simulation débute à partir d'un réseau vide. Les différentes configurations initiales offertes sont listées

dans le fichier de configuration XML « degreeDistribution.xml ». Lors de l'ouverture du logiciel, les configurations décrites sont montées en mémoire. La figure 4.3.1.1 montre ce fichier.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE DEGREE_DISTRIBUTIONS SYSTEM "DegreeDistributions.dtd">

<DEGREE_DISTRIBUTIONS>

        <DISTRIBUTION>
                <NAME>Explicit degree distribution</NAME>
                <IMPLEMENTATION_CLASS>ExplicitDegreeDistrib</IMPLEMENTATION_CLASS>
                <PARAM>Maximum Degree</PARAM>
                <PARAM>Degree / Number of nodes table</PARAM>
        </DISTRIBUTION>

        <DISTRIBUTION>
                <NAME>Random link distribution</NAME>
                <IMPLEMENTATION_CLASS>LinksProportionDistrib</IMPLEMENTATION_CLASS>
                <PARAM>Number of nodes</PARAM>
                <PARAM>percentage of links</PARAM>
        </DISTRIBUTION>

        <DISTRIBUTION>
                <NAME>Normal distribution</NAME>
                <IMPLEMENTATION_CLASS>NormalDegreeDistrib</IMPLEMENTATION_CLASS>
                <PARAM>Number of nodes</PARAM>
                <PARAM>Mean</PARAM>
                <PARAM>Standard deviation</PARAM>
        </DISTRIBUTION>

        <DISTRIBUTION>
                <NAME>Power law distribution</NAME>
                <IMPLEMENTATION_CLASS>PowerLawDegreeDistrib</IMPLEMENTATION_CLASS>
                <PARAM>Number of nodes</PARAM>
                <PARAM>i</PARAM>
        </DISTRIBUTION>

</DEGREE_DISTRIBUTIONS>
```

Figure 4.3.1.1 Fichier de configuration XML des distributions initiales du réseau offertes par NetSim

Dans le fichier de configuration XML chaque configuration est décrite par les balises <NAME>, <IMPLEMENTATION_CLASS> et <PARAM> contenues à l'intérieur de chaque balise <DISTRIBUTION>. Le tableau 4.3.1.1 décrit chacune de ces balises.

Tableau 4.3.1.1 Description des balises du fichier XML de configuration des distributions initiales du réseau offerte par NetSim

Étiquette	Description
<NAME>	Spécifie le nom de la distribution. Ce nom est celui qui apparaîtra dans la liste des distributions offertes dans l'interface graphique de NetSim.
<IMPLEMENTATION_CLASS>	Spécifie la classe d'implémentation de la distribution.
<PARAM>	Spécifie le nom des paramètres que l'utilisateur devra saisir, dans l'interface graphique, s'il choisit cette distribution initiale.

Dans le fichier de la figure 4.3.1.1, quatre distributions initiales sont définies. La première distribution est une distribution explicite pour laquelle l'utilisateur doit tout d'abord spécifier le degré maximum du réseau à générer et ensuite, doit donner le nombre de nœuds, dans le réseau, pour chaque degré allant de zéro au degré maximum. Par exemple, s'il choisit un degré maximum égal à 4, il devra ensuite spécifier le nombre de nœuds de degré égal à 0, le nombre de nœuds de degré égal à 1, le nombre de nœuds de degré égal à 2 et ainsi de suite jusqu'à 4. Le réseau initial de la simulation sera donc initialisé avec ces paramètres.

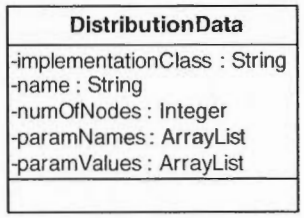
La seconde distribution est une distribution aléatoire pour laquelle l'utilisateur doit spécifier le nombre de nœuds du réseau initial et le pourcentage de liens dans le réseau. Cette configuration est construite en ajoutant d'abord les nœuds, puis les liens, de façon aléatoire.

La troisième distribution décrite est une distribution (des degrés) normale pour laquelle l'utilisateur doit fournir le nombre de nœuds, la moyenne et l'écart-type.

La dernière distribution est une distribution (des degrés) en loi de puissance pour laquelle l'utilisateur doit spécifier le nombre de nœuds et le paramètre i de la formule de la loi de puissance $p(degré) = degré^{-i}$.

Le fichier XML des configurations initiales est automatiquement lu en mémoire à l'initialisation du logiciel. Le nom de chaque configuration initiale, spécifié par la balise

<NAME>, est affiché dans l'interface graphique et lorsque l'utilisateur en choisit une, il doit fournir une valeur pour chacun des paramètres spécifiés par les balises <PARAM>. La configuration choisie, les valeurs des paramètres fournies par l'utilisateur ainsi que la classe d'implémentation de cette distribution, donnée par la balise <IMPLEMENTATION_CLASS>, sont alors stockées en mémoire dans l'objet DistributionData. La figure 4.3.1.2 détaille cette classe.



implementationClass	Le nom de la classe, dans le paquetage org.netsim.application, qui implémente la construction de cette distribution.
Name	Le nom de cette distribution
numOfNodes	Le nombre de nœuds de cette distribution (s'il y a lieu)
paramNames	La liste des noms des paramètres
paramValues	La liste des valeurs des paramètres (dans le même ordre que les noms des paramètres correspondants dans paramNames).

Figure 4.3.1.2 Les attributs de la classe DistributionData
La classe DistributionData contient les informations nécessaires à la construction de la configuration initiale du réseau choisie par l'utilisateur.

Toutes les classes d'implémentation doivent hériter de la classe abstraite DegreeDistribConfig qui contient la méthode abstraite createDistrib(). Les classes d'implémentation doivent alors implémenter la méthode createDistrib() dont le travail est de construire la configuration initiale. Ainsi, lorsque vient le moment de générer le réseau initial, le programme construit d'abord un objet du type de la classe d'implémentation. Le constructeur de chaque classe d'implémentation doit recevoir en argument la liste des valeurs des paramètres de la configuration initiale (paramValues) obtenue de l'objet DistributionData. Le programme invoque ensuite la méthode

`createDistrib()` sur cet objet. La figure 4.3.1.3 montre un diagramme UML des classes qui viennent d'être discutées.

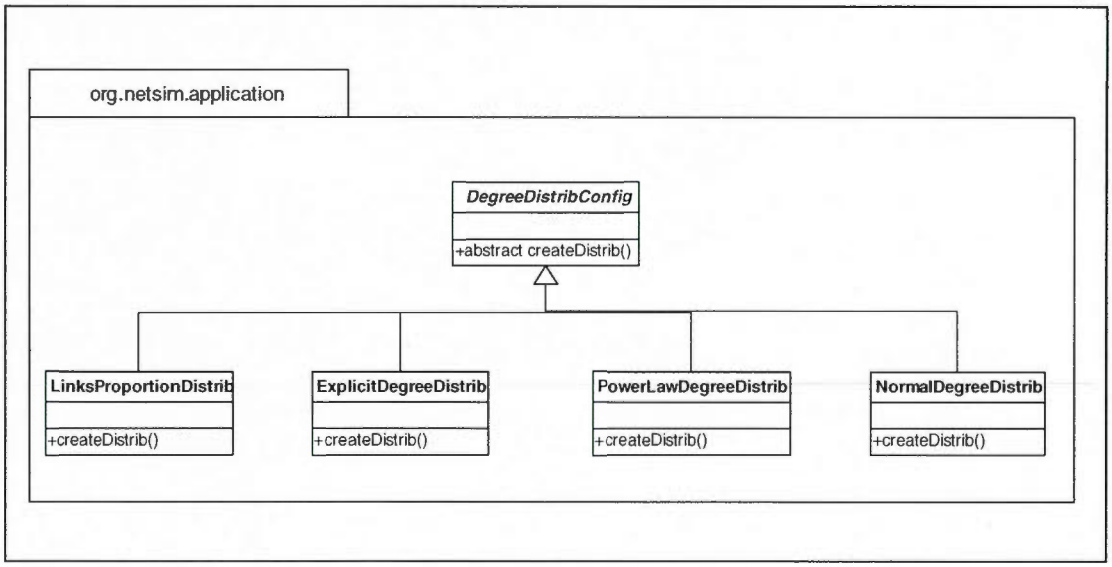


Figure 4.3.1.3 Diagramme UML représentant les classes d'implémentation des configurations initiales du réseau.
Les classes d'implémentation se trouvent dans le paquetage org.netsim.application et doivent toutes hériter de la classe abstraite `DegreeDistribConfig` qui contient la méthode abstraite `createDistrib()`. Par conséquent, les classes d'implémentation doivent aussi implémenter la méthode `createDistrib()`.

Maintenant, en regardant le fichier XML des configurations initiales, on constate qu'il y en a quatre cependant, NetSim a été conçu de telle sorte que l'on puisse aisément en ajouter de nouvelles. Supposons, par exemple, que l'on veuille ajouter le choix d'une distribution suivant une loi de Poisson. La Première étape est d'ajouter les informations nécessaires au fichier XML des configurations initiales. Par exemple :

```
<DISTRIBUTION>
        <NAME>Poisson distribution</NAME>
        <IMPLEMENTATION_CLASS>PoissonDegreeDistrib</IMPLEMENTATION_CLASS>
        <PARAM>Number of nodes</PARAM>
        <PARAM>mean</PARAM>
</DISTRIBUTION>
```

La seconde étape est de créer la classe d'implémentation `org.netsim.application.PoissonDegreeDistrib.java` qui hérite de la classe `DegreeDistribConfig`. Notre classe d'implémentation doit ensuite

implémenter la méthode `createDistrib()` dans laquelle on écrit le code qui génère la distribution suivant une loi de Poisson. Le constructeur de notre classe d'implémentation reçoit en argument une liste des valeurs des paramètres nécessaires à la construction de cette configuration, préalablement saisies par l'utilisateur et se trouvant dans l'objet `DistributionData`. Le code simplifié de la classe d'implémentation pourrait ressembler à ceci :

```
public class PoissonDegreeDistrib extends DegreeDistribConfig {
      ArrayList distribParams;

      //constructeur
      public PoissonDegreeDistrib(ArrayList params) {
            this.distribParams = params;
      }

      public void generateDistrib (int numOfNodes, double mean) {
            //À FAIRE
      }


      //méthode qui génère la distribution
      public void createDistrib () {
            try {
                  //recueillir les paramètres
                  int numOfNodes =
                        (int)((Double)distribParams.get(0)).
                        doubleValue();

                  double mean = ((Double)distribParams.
                        get(1)).doubleValue();

                  //générer la distribution
                  generateDistrib (numOfNodes, mean);

            } catch (ClassCastException e) {
                  System.out.println("A parameter is wrong.");
            }
      }

} //Fin classe
```

Pour générer les distributions des degrés suivant différentes lois comme la loi normale et la loi de puissance, nous avons utilisé des méthodes de génération de nombres provenant de classes Java contenues dans les paquetages cern.colt.* , cern.jet.*, cern.clhep du projet Colt (http://dsd.lbl.gov/~hoschek/colt). Ce projet fournit un ensemble de librairies Java

permettant d'effectuer une grande variété d'opérations et de calculs scientifiques et techniques. Nous reproduisons ici les permissions et les droits d'auteurs.

```
Copyright (c) 1999 CERN - European Organization for Nuclear Research.
Permission to use, copy, modify, distribute and sell this software and
its documentation for any purpose is hereby granted without fee, provided
that the above copyright notice appear in all copies and that both that
copyright notice and this permission notice appear in supporting
documentation. CERN makes no representations about the suitability of
this software for any purpose. It is provided "as is" without
expressed or implied warranty.
```

Implémentation des opérateurs de NetSim

Tout d'abord, un opérateur est soit unaire, soit binaire. NetSim n'implémente pas, pour le moment, d'opérateurs tertiaires. Ensuite, il peut être un opérateur de comparaison, arithmétique, booléen ou d'affectation. Le fichier « operatorsDefinitions.txt » contient le classement de tous les opérateurs de NetSim. La figure 4.3.1.4 montre ce fichier. La première ligne du fichier sert à définir tous les symboles atomiques utilisés (en combinaison ou non) pour représenter les opérateurs symboliques (qui ne sont pas des mots). Les lignes suivantes classent les opérateurs dans leurs catégories. Par exemple, l'opérateur « := » est un opérateur binaire d'affectation tandis que l'opérateur « floor » est unaire et arithmétique.

```
symbols     ->    +,-,*,/,=,:,<,>,\
unary       ->    not,floor,-,+
binary      ->    :=,=,>,<,>=,<=,/=,+,-,*,/,**,mod,and,or,xor
assignment  ->    :=
comparison  ->    =,>,<,>=,<=,/=
arithmetic  ->    +,-,*,/,**,floor,mod
boolean     ->    and,or,not,xor
```

Figure 4.3.1.4 Fichier de classement des opérateurs de NetSim

Comme nous l'avons expliqué à la section 4.2, les déclarations en langages NetSim sont d'abord traduites en langage Java avant d'être compilées et exécutées. Nous avons donc deux autres fichiers qui indiquent la correspondance entre les opérateurs de NetSim et les

opérateurs Java. La figure 4.3.1.5 illustre ces deux fichiers. Le fichier de gauche fait correspondre les opérateurs de NetSim avec les opérateurs de Java tandis que celui de droite fait correspondre les opérateurs de NetSim avec des méthodes Java contenues dans la classe java.lang.Math.

```
not   ->   !
and   ->   &&
or    ->   ||
xor   ->   ^
mod   ->   %
:=    ->   =
=     ->   ==
/=    ->   !=
```

```
**    -> Math.pow
floor -> Math.floor
```

Figure 4.3.1.5 Fichiers de correspondance des opérateurs de NetSim avec les opérateurs ou les méthodes Java
Le fichier de gauche contient les correspondances des opérateurs de NetSim avec les opérateurs de Java. Le fichier de droite contient les correspondances des opérateurs de NetSim avec les méthodes Java de la classe java.lang.Math.

Ces fichiers serviront à la traduction des expressions arithmétiques ou booléennes de NetSim en expression Java. Cette opération est expliquée plus en détail à la section 4.3.2.

Maintenant, il est possible d'inventer ou d'ajouter d'autres opérateurs NetSim en modifiant ces trois fichiers de configuration. Les deux exemples suivants montrent comment faire.

Exemple 1

Supposons que l'on veuille ajouter l'opérateur de comparaison « >> » qui signifie « plus de deux fois plus grand ».

Par exemple, `x >> y` signifie la même chose que `x > 2 * y`.

Dans le fichier de classement des opérateurs :

Vérifier que le symbole atomique « > » utilisé pour former notre nouvel opérateur « >> » est présent dans la première ligne. Dans cet exemple, il y est, mais dans le cas où il n'y serait pas, il faudrait l'ajouter à la fin de la première ligne du fichier en le séparant des autres symboles par une virgule.

Ensuite, puisque notre nouvel opérateur est un opérateur de comparaison binaire, il faut l'ajouter à la fin des lignes suivantes en le séparant des autres opérateurs par une virgule :

```
binary        ->     :=,=,>,<,>=,<=,/=,+,,*,/,**,mod,and,or,xor,>>
comparison    ->    =,>,<,>=,<=,/=,>>
```

Dans le fichier de correspondance des opérateurs de NetSim avec les opérateurs Java :

Ajouter une ligne au fichier de gauche, dans la figure 4.3.1.5, en indiquant la traduction de l'opérateur comme ceci :

```
>>     ->    > 2 *
```

Ainsi, lors de la traduction, l'opérateur NetSim « >> » sera transformé par l'expression Java « `2 * >` ».

Puisque l'expression Java ne fait pas appel à une méthode de la classe java.lang.Math, nous ne modifions pas le fichier de droite de la figure 4.3.1.5.

Exemple 2 :

Supposons que l'on désire ajouter la fonction sqrt qui calcule la racine carrée d'un nombre de telle sorte que l'expression « `sqrt 4` » égale 2.

Dans le fichier de classement des opérateurs :

Étant donné que notre nouvel opérateur « `sqrt` » n'est pas formé de symboles atomiques (c'est un mot), on ne vérifie pas la première ligne du fichier.

Notre nouvel opérateur est un opérateur unaire arithmétique. Il faut donc ajouter cet opérateur à la fin des lignes qui comptabilisent ces types d'opérateurs, de cette façon :

```
unary       ->    not,floor,-,+,sqrt
arithmetic  ->    +,-,*,/,**,floor,mod,sqrt
```

Dans le fichier de correspondance des opérateurs de NetSim avec les méthodes Java de la classe java.lang.Math :

Ajouter la ligne suivante :

```
sqrt -> Math.sqrt
```

Avec ces simples modification des fichiers de configuration des opérateurs, il est maintenant possible d'utiliser l'opérateur « >> » et l'opérateur « `sqrt` » dans la définition des fonctions ou des paramètres des actions lors de l'écriture du modèle en langage de modélisation NetSim.

Implémentation des fonctions prédéfinies sur chaque entité

Comme nous l'avons vu au chapitre 3 sur le langage de modélisation NetSim, il existe des fonctions prédéfinies sur chaque entité. Ces fonctions correspondent à des méthodes Java qui sont implémentées dans la classe `SimulationController`. Par exemple, la fonction `number_of_nodes()`, définie sur l'entité network, correspond à la méthode `getNumberOfNodes()` de la classe `SimulationController`. NetSim possède un tel fichier de correspondance par entité. Par exemple, le fichier « networkFunctions.properties », illustré à la figure 4.3.1.6 contient la correspondance entre les fonctions NetSim définies sur l'entité network et les méthodes Java qui les implémentent.

```
number_of_nodes:getNumberOfNodes()
number_of_links:getNumberOfLinks()
mean_degree:getMeanDegree()
max_degree:getMaxDegree()
min_degree:getMinDegree()
diameter:getDiameter()
```

Figure 4.3.1.6 Fichier de correspondance entre les fonctions NetSim définies sur l'entité network et les méthodes Java qui les implémentent

À l'initialisation du logiciel, ces fichiers sont lus en mémoire et serviront à la traduction des expressions NetSim en expression Java.

Pour ajouter une nouvelle fonction prédéfinie sur une entité particulière, il suffit donc d'implémenter la méthode Java correspondante dans la classe `SimulationController` et d'ajouter la correspondance dans le fichier de l'entité concerné.

Il existe cependant certaines restrictions aux fonctions qui peuvent être ajoutées au logiciel. Pour le moment, il n'est pas possible d'implémenter des fonctions avec paramètres, mais nous comptons éventuellement remédier à cette limitation.

Les codes de couleur

Dans le langage de modélisation NetSim, nous avons vu qu'il existe des attributs de couleur prédéfinis sur les entités node et link. Ces attributs permettent de modifier la couleur des nœuds et des liens du réseau en cours de simulation. Il faut pour cela affecter à l'attribut de couleur un code numérique correspondant à une couleur définie dans le fichier de configuration « colorCodes.properties ». Chaque ligne de ce fichier contient la définition d'une couleur en précisant le code NetSim de la couleur, son encodage RVB et finalement, son nom. Par exemple, pour définir la couleur rouge, nous pouvons écrire, dans le fichier, la ligne suivante :

```
7:255,0,0;light red
```

La ligne précédente définit la couleur rouge dont le code NetSim est `7`, l'encodage RVB est `255, 0, 0` et le nom est `light red`. Il faut bien entendu que les codes de couleur NetSim soient uniques, dans le fichier.

On peut ainsi ajouter toutes les couleurs voulues au fichier de configuration des couleurs et ces codes seront disponibles pour modifier les attributs de couleur définis sur les entités node et link.

4.3.2 Traduction du modèle NetSim en modèle Java

À l'aide des données provenant des fichiers d'initialisation de l'application et du `ModelDefinitions` qui contient les définitions du modèle définies par l'utilisateur, NetSim peut traduire les règles du modèle.

Après que l'`ApplicationController` ait appelé sa méthode `loadModel()` pour construire le `SimulationModel`, il appelle sa méthode `initJavaSimulationModel()` pour construire le modèle Java en mémoire. Pour ce faire, il traduit chaque expression NetSim du `SimulationModel` : la condition d'arrêt de la simulation, les conditions d'exécution de chaque phase et les règles (actions) faisant partie de chaque phase. Pour chaque expression, il construit un objet `NetSimExpression` en lui passant en paramètre l'expression NetSim à traduire.

C'est la méthode `netSimToJava()`, appelée sur l'objet `NetSimExpression` qui est responsable de la traduction des expressions NetSim en expressions Java. Tout d'abord, elle traduit les fonctions, ensuite, les nombres, les constantes, les attributs, les opérateurs et finalement, les affectations (qui sont aussi des opérateurs, mais qui sont traitées différemment).

Prenons, par exemple, l'expression suivante qui est celle de la fonction `g` de notre modèle d'amis :

```
1 - ( 1 - p0 ) * e ** ( - a * number_of_common_contacts() )
```

La première étape de la méthode `netSimToJava()` consiste à traduire les fonctions. Pour ce faire, l'application peut vérifier quelles sont les fonctions prédéfinies de NetSim, dans l'`ApplicationDataRegister` et quelles sont les fonctions définies par l'utilisateur dans le `ModelDefinitions`. Après la traduction des fonctions, l'expression ressemble à ceci :

```
1 - ( 1 - p0 ) * e ** ( - a * simController.getNumOfCommonNeighbours
(link) )
```

La deuxième étape consiste à traduire tous les nombres en nombres réels pour éviter les divisions entières. Après la traduction des nombres, l'expression ressemble à ceci :

```
1.0 - ( 1.0 - p0 ) * e ** ( - a *
simController.getNumOfCommonNeighbours (link) )
```

La troisième étape consiste à traduire les constantes dont les valeurs peuvent être récupérées dans le `ModelDefinitions`. Après la traduction des constantes, l'expression ressemble à ceci :

```
1.0 - ( 1.0 - 0.0060 ) * 2.718 ** ( - 0.5 *
simController.getNumOfCommonNeighbours (link) )
```

La quatrième étape consiste à remplacer le nom des attributs par la méthode `getAttribute (Entité, nomAttrib)` du `SimulationController` qui récupère la valeur courante de cet attribut au moment de l'appel. Dans cet exemple, il n'y a aucun attribut donc l'expression reste inchangée.

La cinquième étape est la traduction des opérateurs, à l'exception de l'opérateur d'affectation qui sera traduit à l'étape suivante. Toutes les indications nécessaires à la traduction des opérateurs proviennent des fichiers d'initialisations des opérateurs lus en mémoire dans l'objet `ApplicationDataRegister`. Après la traduction des opérateurs, l'expression ressemble à ceci;

```
1.0 - ( 1.0 - 0.0060 ) * Math.pow (2.718, - 0.5 *
simController.getNumOfCommonNeighbours (link))
```

La sixième et dernière étape est la traduction des opérateurs d'affectation. Cet opérateur est traduit par la méthode `setAttribute (entité, nomAttrib, nouvelleValeur)` du `SimulationController`. Dans cette expression, il n'y a pas d'opérateur d'affectation alors l'expression reste inchangée.

L'expression finale, complètement traduite en code Java, est la suivante :

```
1.0 - ( 1.0 - 0.0060 ) * Math.pow (2.718, - 0.5 *
simController.getNumOfCommonNeighbours (link))
```

Comme exemple de traduction de l'opérateur d'affectation et des attributs, prenons l'expression suivante, aussi tirée du modèle d'amis :

```
strength := s()
```

Après la traduction des fonctions :

```
strength := ( e ** ( - k * time_since_last_meeting ) )
```

Après la traduction des nombres :

```
strength := ( e ** ( - k * time_since_last_meeting ) )
```

Après la traduction des constantes :

```
strength := ( 2.718 ** ( - 0.01 * time_since_last_meeting )
```

Après la traduction des attributs :

```
strength := ( 2.718 ** ( - 0.01 * simController.getAttribute(link,
"time_since_last_meeting") ) )
```

Après la traduction des opérateurs :

```
strength := ( Math.pow(2.718, - 0.01 * simController.getAttribute
(link, "time_since_last_meeting")) )
```

Après la traduction de l'affectation :

```
simController.setAttribute (link, "strength", ( Math.pow (2.718, -
0.01 * simController.getAttribute (link, "time_since_last_meeting"))
))
```

Maintenant que toutes les expressions sont traduites en expression Java, l'application peut écrire les classes d'exécution de la simulation. C'est ce que nous expliquons à l'article suivant.

4.3.3 Écriture des classes Java d'exécution de la simulation

La classe principale d'exécution de la simulation est la classe `SimulationContext`. Elle contient la méthode `run()` dans laquelle se trouve la boucle principale d'exécution de la simulation. Un tour de boucle correspond à un pas de temps et à chaque pas de temps s'exécutent une ou plusieurs phases selon le mode d'exécution (séquentiel ou parallèle) choisi par l'utilisateur. La figure 4.3.3.1 montre les classes nécessaires à l'exécution d'une simulation.

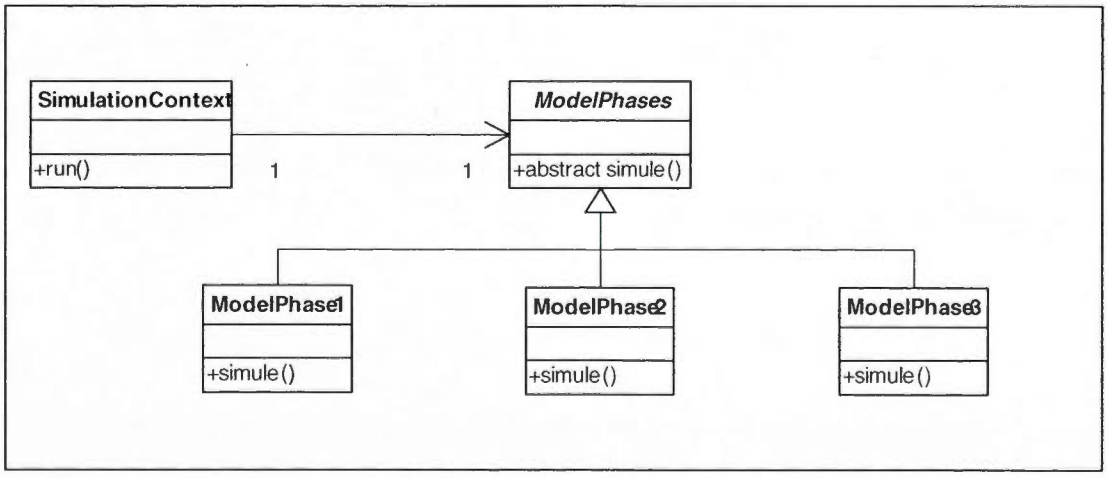


Figure 4.3.3.1 Diagramme de classes UML représentant les classes d'exécution de la simulation

Premièrement, il y la classe `SimulationContext` dont nous venons de parler. Ensuite, chaque phase du modèle correspond à une classe `ModelPhase#` (où # représente le numéro de la phase). Chaque classe `ModelPhase#` hérite de la classe abstraite `ModelPhases` qui contient la méthode abstraite `simule()`. Toutes les classes `ModelPhase#` doivent donc implémenter cette méthode.

La méthode `simule()` de chaque `ModelPhase#` est responsable de l'exécution des actions contenues dans cette Phase. Voici, comme exemple, une représentation générale simplifiée de la classe `ModelPhase1` qui contient, disons, une action `ADD_NODES`, une action `PICK_LINKS` ,une action `ADD_LINKS` et une autre action `ADD_NODES` :

```
public class ModelPhase1 extends ModelPhases {

      public void addNodes0 (SimulationController simController) {
            //code de la première action add_nodes
      } //end method

      public void pickLinks1 (SimulationController simController) {
            //code de la deuxième action pick_links
      } //end method

      public void addLinks2 (SimulationController simController) {
            //code de la troisième action add_links
      } //end method

      public void addNodes3 (SimulationController simController) {
            //code de la dernière action add_nodes
      } //end method


      //implémentation de la méthode simule
      public void simule() {

      SimulationController simController =
                  SimulationController.getController();

            addNodes0   (simController);
            pickLinks1  (simController);
            addLinks2   (simController);
            addNodes3   (simController);

      } //end method
} //end class
```

Dans chaque classe `ModelPhase#`, une action est implémentée par une méthode du nom de cette action qui prend en paramètre une instance de la classe `SimulationController`. En effet, lors de la traduction des expressions NetSim en Java, toutes les méthodes sont appelées sur cet objet.

Évidemment, le code de chaque méthode qui implémente une action particulière sera différent selon les paramètres que l'utilisateur a fournis à cette action lors de l'écriture du modèle en langage de modélisation NetSim. Ce sont ces méthodes qui modifient le graphe représentant le réseau simulé. Nous ne verrons pas le code de ces méthodes en détail cependant, l'annexe A fournit le code complet des classes `SimulationContext` et des classes `ModelPhase#` écrites dynamiquement lors de la construction de la simulation du modèle de réseaux d'amis.

Comme nous l'avons vu à la section 3.5.2, le code de la classe `SimulationContext` sera différent selon que le mode d'exécution choisi par l'utilisateur est séquentiel ou parallèle.

Revoyons le pseudo-code général de la méthode `run()` selon ces deux modes (en supposant qu'il y a trois phases dans le modèle et que la première phase n'a pas de condition d'exécution):

Mode d'exécution séquentiel

```
compteurDePhases = 0
pasDeTemps = 0
ModelPhases p = null

TANT QUE NOT <condition d'arrêt de la simulation> FAIRE

      # phase 3
      SI <condition d'exécution phase 3> et compteurDePhases = 2
      ALORS
            compteurDePhases = 3
            p := nouvelle instance de la classe ModelPhase3

      # phase 2
      SINON SI <condition d'exécution phase 2> et
      compteurDePhases = 1 ALORS
            compteurDePhases = 2
```

```
            p := nouvelle instance de la classe ModelPhase2

      # phase 1
      SINON SI compteurDePhases = 0 ALORS
            compteurDePhases = 1
            p := nouvelle instance de la classe ModelPhase1

      FIN SI

      p.simule()

      pasDeTemps = pasDeTemps + 1

FIN TANT QUE
```

Mode d'exécution parallèle

```
pasDeTemps = 0

TANT QUE NOT <condition d'arrêt de la simulation> FAIRE

      p := nouvelle instance de la classe ModelPhase1
      p.simule()

      SI <condition d'exécution phase 2> ALORS
            p := nouvelle instance de la classe ModelPhase2
            p.simule()

      FIN SI

      SI <condition d'exécution phase 3> ALORS
            p := nouvelle instance de la classe ModelPhase3
            p.simule()

      FIN SI

      pasDeTemps = pasDeTemps + 1

FIN TANT QUE
```

Dans les deux cas, si une phase doit être exécutée, une instance de la classe `ModelPhase#` correspondante est créée et la méthode `simule()` est appelée sur cette instance.

Les classes qui sont écrites en mémoire sont donc la classe `SimulationContext` et toutes les classes `ModelPhase#`. La classe `ModelPhases` n'est pas écrite dynamiquement étant donné qu'elle ne change pas selon les modèles simulés.

Les chaînes de caractères qui représentent la définition de chacune de ces classes sont entreposées dans le `ModelDefinitions`.

4.3.4 Compilation de la simulation

Le choix de conception que nous avons fait d'utiliser la machine virtuelle Java comme interpréteur de notre langage de modélisation NetSim préalablement traduit en Java suppose la compilation dynamique des classes discutées à l'article précédent. Nous nous sommes interrogés à savoir si la compilation de classes ne prendrait pas trop de temps lors de la mise en œuvre d'une simulation, mais étant donné que le nombre de classes à compiler est proportionnel au nombre de phases du modèle simulé, qui est rarement plus grand que trois, la compilation s'exécute très rapidement.

Lorsque les classes d'exécution de la simulation ont été écrites en mémoire et stockées dans le `ModelDefinitions`, nous pouvons les compiler. Pour ce faire, nous avons utilisé le paquetage uk.ac.dcs.stand.dcs.javacompiler créé par Graham Kirby (http://www-systems.cs.st-andrews.ac.uk/wiki/Dynamic_Java_Compiler), contenant une librairie pour la compilation dynamique.

C'est la méthode `compileSimulation()` de la classe `ApplicationController` qui s'occupe de la compilation des classes d'exécution de la simulation. Il suffit de fournir au compilateur dynamique le nom de la classe que l'on veut compiler ainsi qu'un tableau contenant les définitions de cette classe et de toutes les autres classes dépendantes. Le compilateur retourne une instance de la Classe Class de la classe compilée. Dans notre cas, nous fournissons le nom de la classe « SimulationContext » et un tableau des définitions des classes `SimulationContext` et toutes les classes `ModelPhase#`. Le code Java suivant, tiré de la méthode `compileSimulation()` de l'`ApplicationController` montre les étapes principales de la compilation.

```
//Definitions de variables
String prefix = SimulationConst.SIMULATION_PACK_PREFIX;
ModelDefinitions modelDefs = simRegister.getModelDefinitions();
String [] defns = modelDefs.getClassesDefinitions();
Class simulContextClass = null;
DynamicCompiler compiler = null;

//créer le compilateur dynamique
compiler = new DynamicCompiler();

//compilation des classes
simulContextClass = compiler.compileClass (defns, prefix +
    "SimulationContext");

//stockage de la classe compilée dans le ModelDefinitions
modelDefs.setCompiledSimContextClass(simulContextClass);

//Création d'une instance de la classe compilée et stockage de cette
//instance dans le ModelDefinitions
modelDefs.setSimContext(simulContextClass.newInstance());
```

La variable `prefix` contient le nom du paquetage dans lequel se trouve la classe `SimulationContext`. Le tableau `defns` est un tableau de chaînes de caractères contenant les définitions des classes à compiler, préalablement stockées dans le `ModelDefinitions`.

La méthode `compile()` de l'objet `compiler` retourne l'objet `simulContextClass` (une instance de la classe `Class`) qui est alors stocké dans le `ModelDefinitions`. Cet objet contient le code octet de la classe `SimulationContext`.

Finalement, en appelant la méthode `newInstance()` sur l'objet `simulContextClass`, nous construisons une instance de la classe `SimulationContext` et cette instance est aussi stockée dans le `ModelDefinitions`.

4.3.5 Exécution de la simulation

Comme nous l'avons déjà dit, c'est le `SimulationController` qui est responsable du contrôle de l'exécution de la simulation. Sa méthode `startSimulation()` démarre la

simulation en invoquant la méthode `run()` sur l'instance de la classe `SimulationContext` que nous avons préalablement compilée et stockée dans le `ModelDefinitions`.

Le `SimulationController` contient aussi la méthode `stopSimulation()` qui termine la simulation, la méthode `pauseSimulation()` qui permet l'arrêt temporaire de la simulation ainsi que la méthode `resumeSimulation()` qui permet la reprise de l'exécution de la simulation après un arrêt temporaire. La classe `SimulationContext` est en fait une tâche (thread) qui peut donc être suspendue et redémarrée à volonté.

4.3.6 Données de réseaux externes

À la section 4.3.1, nous avons introduit la notion de configuration initiale en expliquant que NetSim offre différentes façons de créer le réseau initial de la simulation selon différentes distributions. Cette fonctionnalité permet à l'utilisateur de ne pas amorcer sa simulation à partir d'un réseau vide.

Cependant, il peut arriver que l'on possède déjà des données de réseau que l'on voudrait utiliser pour construire le réseau qui servira de point de départ à notre simulation. Dans cette optique, nous avons créé un format de fichier XML permettant la représentation de données de réseaux externes et que NetSim peut alors interpréter pour créer le réseau initial. Il suffit d'écrire nos données sous ce format et de charger le fichier XML dans NetSim.

Aussi, NetSim permet d'enregistrer dans ce format les réseaux générés par simulation. Cela peut être utile dans le cas où nous voudrions récupérer nos données dans le but de les utiliser dans un autre logiciel, d'analyse par exemple ou simplement pour les réutiliser comme réseau initial dans une autre simulation de NetSim.

Le format XML de NetSim (NetSimXML) est divisé en deux parties. La première, délimitée par la balise `<ENTITY_ATTRIBUTE>` sert à définir le nom des attributs des nœuds et des liens. S'il n'y a pas d'attributs, cette section reste vide. La seconde partie, délimitée par la

balise <GRAPH> sert à décrire tous les nœuds et les liens du graphe avec les valeurs de leurs attributs respectifs s'il y a lieu. La figure 4.3.6.1 montre un exemple d'un tel fichier.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE NETWORK SYSTEM "externalData.dtd">

<NETWORK>

    <ENTITY_ATTRIBUTES>
        <ENTITY_ATTRIBUTES>
                <NODE_ATTRIB>
                    <NAME>attrib1</NAME>
                    <NAME>attrib2</NAME>
                </NODE_ATTRIB>

                <LINK_ATTRIB>
                        <NAME>attrib3</NAME>
                        <NAME>attrib4</NAME>
                </LINK_ATTRIB>
    </ENTITY_ATTRIBUTES>

    <GRAPH oriented="true">

        <NODES>
                <NODE id="n1"  label="Banane" attrib_values="2.3 3" />
                <NODE id="n2" label="Maison" attrib_values="0.345 3" />
                <NODE id="n3" label="Citron" attrib_values="2.5 3.2" />
                <NODE id="n4" label="Café" attrib_values="2.6 3.5" />
                <NODE id="n5" label="Taxi" attrib_values="0 0" />
                <NODE id="n6" label="Voiture" attrib_values="2.6 3.5" />
                <NODE id="n7" label="Soleil" attrib_values="2.6 3.5" />
                <NODE id="n8" label="Visage" attrib_values="2.6 3.5" />
                <NODE id="n9" label="Beau" attrib_values="2.5 3.2" />
                <NODE id="n10" label="Travail" attrib_values="0 0" />
                <NODE id="n11" label="École" attrib_values="2.5 3.2" />
                <NODE id="n12" label="Moustache" attrib_values="2.5 3.2" />
                <NODE id="n13" label="Lunette" attrib_values="2.5 3.2" />
        </NODES>

        <LINKS>
                <LINK node1="n1"  node2="n12" attrib_values="0 1"/>
                <LINK node1="n3"  node2="n7" attrib_values="2 0" />
                <LINK node1="n6"  node2="n7" attrib_values="3 1" />
                <LINK node1="n12" node2="n3" attrib_values="0 2" />
                <LINK node1="n1"  node2="n4" attrib_values="1 3" />
                <LINK node1="n8"  node2="n3" attrib_values="0 3" />
                <LINK node1="n9"  node2="n6" attrib_values="2 0" />
                <LINK node1="n6"  node2="n2" attrib_values="0 1" />
                <LINK node1="n9"  node2="n1" attrib_values="0 1" />
                <LINK node1="n12" node2="n6" attrib_values="0 2" />
                <LINK node1="n12" node2="n5" attrib_values="0 0" />
                <LINK node1="n4"  node2="n12" attrib_values="2 0" />
        </LINKS>
    </GRAPH>
</NETWORK>
```

Figure 4.3.6.1 Données de réseaux externes dans le format NetSimXML

On remarque que le format NetSimXML permet aussi de spécifier une étiquette pour chaque nœud du graphe, mais celle-ci n'est pas obligatoire. Si on mentionne une étiquette dans la définition des noeuds, elles seront affichées dans chaque nœud lors de la visualisation du graphe.

Dans l'exemple de la figure 4.3.6.1, on a défini deux attributs pour les nœuds (`attrib1` et `attrib2`) ainsi que deux attributs pour les liens (`attrib3` et `attrib4`). Les valeurs de ces attributs sont spécifiées dans la définition de chaque nœud et de chaque lien. Ainsi, la ligne

```
<NODE id="n1"  label="Banane" attrib_values="2.3 3" />
```

définit un nœud dont le numéro unique d'identification est « n1 », l'étiquette est « Banane », la valeur de l'attribut `attrib1` est 2.3 et la valeur de l'attribut `attrib2` est 3, les valeurs de chaque attribut étant séparées par un espace. Il est important que les numéros d'identification des nœuds soient tous uniques puisqu'ils serviront à identifier les deux nœuds qui composent un lien dans la définition des liens. Ainsi, la ligne

```
<LINK node1="n1"  node2="n12" attrib_values="0 1"/>
```

définit un lien dont le premier nœud est celui qui a le numéro d'identification « n1 » et dont le deuxième nœud est celui qui a le numéro d'identification « n12 ». La valeur de l'attribut `attrib3` est égale à 0 et la valeur de l'attribut `attrib4` est égale à 1, les valeurs de chaque attribut étant séparées par un espace.

La figure 4.3.6.2 montre le graphe obtenu une fois que le fichier NetSimXML de la figure 4.3.6.1 a été chargé dans NetSim comme configuration de départ.

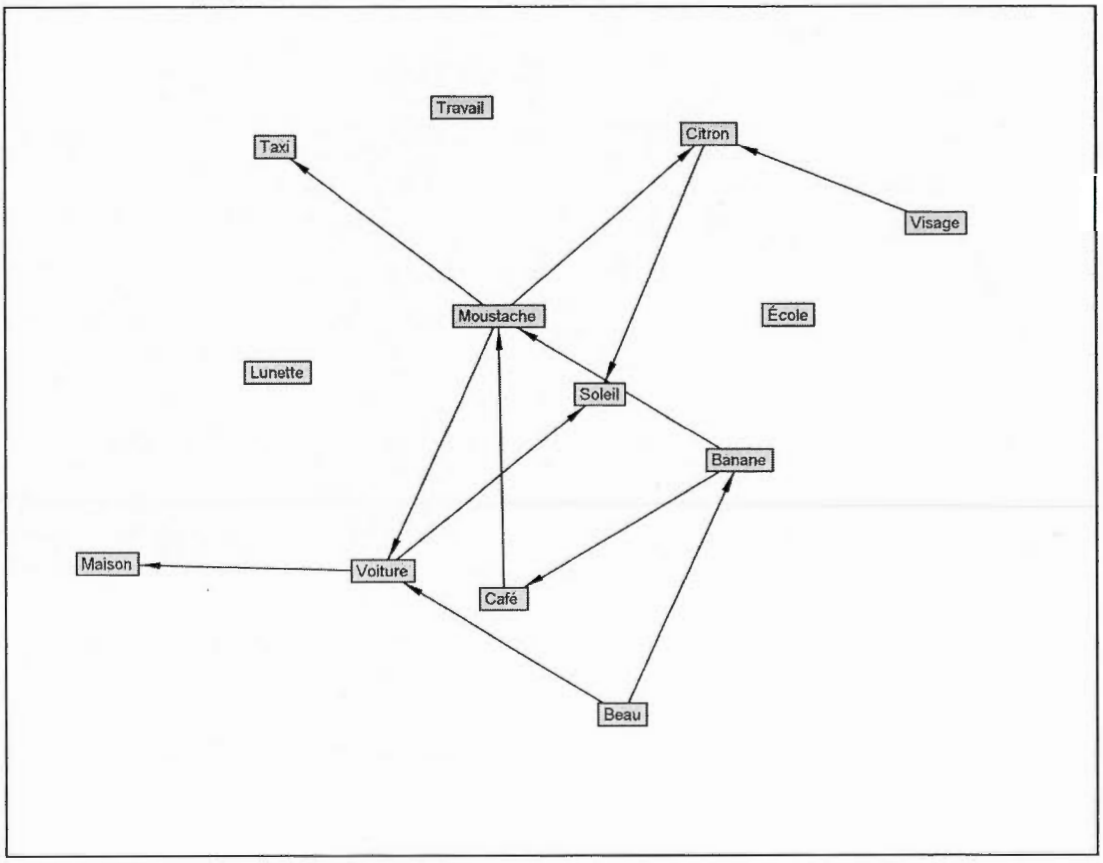


Figure 4.3.6.2 Visualisation de données de réseau externes dans NetSim

Nous pouvons maintenant exécuter une simulation à partir de ce réseau initial. Cependant, si des nœuds sont ajoutés en cours de simulation, ceux-ci sont anonymes et ne seront évidemment pas étiquetés.

Même si NetSim a été conçu spécialement pour simuler des modèles dynamiques de réseaux d'information, on peut aussi l'utiliser pour la visualisation statique de données de réseaux représentées sous le format NetSimXml. On charge d'abord ce fichier dans NetSim, on crée ensuite une simulation vide (sans aucune définition ni règle) et on choisit le fichier NetSimXML comme configuration de départ de notre simulation. Lors de l'initialisation de la simulation, le réseau initial est affiché.

Éventuellement, nous voudrions rendre NetSim capable de lire d'autres formats de représentation de données de réseaux. Par exemple, il serait utile de pouvoir fournir, en format texte ou Excel, des données de réseaux qui représentent simplement les nœuds et les liens sous forme de matrice. Plusieurs applications d'analyse ou de visualisation de réseaux, comme Pajek ou KrackPlot, fournissent leur propre format de données, mais il existe aussi des formats connus et assez génériques, dont GraphXML, qu'il serait intéressant d'intégrer à NetSim.

4.4 Conclusion

Dans ce chapitre, nous avons expliqué le fonctionnement général de NetSim pour ensuite détailler certains aspects de son implémentation. Nous avons pointé, ici et là, certaines limitations actuelles quant à son utilisation, mais avons tenté de montrer que sa conception est assez souple pour permettre, éventuellement, l'extension ou l'amélioration de ses fonctionnalités.

Le chapitre suivant propose deux exemples de simulations de modèles de réseaux tirés de la littérature qui nous ont permis de tester notre plate-forme.

CHAPITRE 5

Résultats de simulation avec NetSim

5.1 Introduction

Lorsque nous étudions certains phénomènes par modélisation, la simulation de nos modèles est une étape essentielle. En effet, celle-ci permet de tester nos hypothèses sous différentes contraintes. Les modèles incluent souvent des paramètres ajustables qu'on doit tester avec différentes valeurs avant de trouver la valeur ou l'intervalle de valeurs qui produit les effets recherchés. De plus, en jouant avec ces paramètres, on peut découvrir des comportements qui n'étaient pas prévus à l'origine. Les simulations permettent ainsi d'expliciter nos modèles.

Dans ce chapitre, nous exposons nos résultats de simulation avec NetSim de deux modèles tirés de la littérature : le modèle du réseau d'amis que nous connaissons déjà et un modèle de l'évolution de la structure du Web. Nous comparons ensuite nos résultats avec les résultats de simulation obtenus par les auteurs de chaque modèle.

5.2 Simulation d'un modèle de l'évolution d'un réseau d'amis

5.2.1 Le modèle théorique

Nous allons reprendre ici le modèle de l'évolution du réseau d'amis de (Jin, Girvan et Newman, 2001) que nous avons déjà écrit au chapitre 3 et allons le simuler à l'aide de NetSim.

Nous en rappelons brièvement ici les principales règles :

1. **Un nombre fixe de sommets** : Considération d'une population fermée d'une grandeur fixe.

2. **Un degré limité** : Lorsqu'un individu atteint z^* contacts, la probabilité que celui-ci développe de nouvelles amitiés diminue radicalement. Ce mécanisme est mis en place pour modéliser le fait qu'entretenir une amitié demande des ressources et que, pour cette raison, un individu ne peut pas avoir un nombre illimité d'amis.

3. **Transitivité** : La probabilité que deux individus deviennent amis doit être significativement plus élevée s'ils ont un ou plusieurs amis communs.

   La probabilité par unité de temps p_{ij} que deux personnes données, i et j, se rencontre dépend de deux facteurs :

   i. Le nombre d'amis z_i et z_j que i et j possèdent déjà.

   ii. Le nombre m_{ij} d'amis communs de i et j.

   Ces facteurs sont représentés par les fonctions f et g :

$$p_{ij} = f(z_i)f(z_j)g(m_{ij})$$

   La fonction f doit être assez grande et assez constante pour de petits z, mais doit diminuer considérablement lorsque z approche la valeur de transition z^*. La fonction de Fermi, qui est une sigmoïde, présente ces caractéristiques et c'est elle que les auteurs ont utilisée pour leur modèle.

$$f(z_i) = \frac{1}{e^{\beta(z_i - z^*)} + 1}$$

   Où le paramètre β contrôle la précision de la diminution à z^*.

La fonction *g(m)* représente l'augmentation attendue de la chance que deux individus se rencontrent s'ils ont un ou plusieurs amis communs.

$$g(m) = 1 - (1 - p_0)e^{-\alpha m}$$

Où p_0 représente la probabilité d'une rencontre entre deux personnes qui n'ont pas d'amis communs et où le paramètre α contrôle la vitesse à laquelle $g(m_{ij})$ augmente.

4. **Bris de liens d'amitié** : Lorsque deux individus sont amis, ceux-ci doivent se rencontrer régulièrement pour maintenir leur amitié. Si tel n'est pas le cas, il y a cessation de cette amitié. Pour représenter la force d'une amitié, chaque lien d'amitié possède un paramètre *s* dont la valeur est plus ou moins élevée selon que les rencontres sont plus ou moins fréquentes. Chaque fois que deux amis se rencontrent, la force s_{ij} est ajustée à la valeur 1. Puis, à mesure que le temps passe et qu'ils ne se rencontrent pas de nouveau, la force de leur amitié diminue exponentiellement selon :

$$s_{ij} = e^{-k\Delta t}$$

Où Δt représente l'intervalle de temps depuis la dernière rencontre de *i* et *j* et où *k* est un paramètre ajustable du modèle.

Si *i* et *j* se rencontre une autre fois, s_{ij} est remis à 1.

Il est ensuite possible de poser une limite minimale à s_{ij} permettant de considérer une amitié active et ne tenir compte que de ces liens lors des calculs, par exemple, du nombre d'amis communs et du coefficient de transitivité.

5.2.2 Le modèle traduit en langage NetSim

Nous avons déjà écrit ce modèle en langage NetSim au chapitre précédent :

Les définitions du modèle en langage NetSim

```
DEFINE_CONST (z_limite, 5);
DEFINE_CONST (b, 5);
DEFINE_CONST (a, 0.5);
DEFINE_CONST (p0, 0.006);
DEFINE_CONST (k, 0.01);
DEFINE_CONST (limit_min_for_friendship, 0.5);
DEFINE_CONST (e, 2.718);

DEFINE_ATTRIB (link, strength, 1);
DEFINE_ATTRIB (link, time_since_last_meeting, 0);

DEFINE_FUNCTION (link, g)
1 - (1 - p0) * e ** (-a * number_of_common_contacts());

DEFINE_FUNCTION (link, fi)
1 /(e ** (b * (start_node().degree() - z_limite)) + 1);

DEFINE_FUNCTION (link, fj)
1 /(e ** (b * (end_node().degree() - z_limite)) + 1);

DEFINE_FUNCTION (link, pij)
g()*fi()*fj();
DEFINE_FUNCTION (link, s)
e ** (-k * time_since_last_meeting);
```

Les règles du modèle en langage NetSim

```
SIMUL_STOP_CONDITION (time_step () >= 1000);

PICK_LINKS(time_since_last_meeting := time_since_last_meeting + 1,
           strength := s());

PICK_LINKS(pij(), time_since_last_meeting := 0 ,
                strength := 1);

CUT_LINKS (strength <= limit_min_for_friendship);

ADD_LINKS(pij());
```

Maintenant que nous avons traduit le modèle en langage NetSim, nous pouvons le simuler.

5.2.3 Résultats de simulation

Pour construire le réseau initial de notre simulation, nous avons fait comme les auteurs du modèle. Nous avons commencé avec 250 nœuds et, en simulant le modèle, nous avons ajouté des liens au réseau jusqu'à ce que le degré moyen des nœuds du réseau atteigne la valeur de la constante `z_limite`. Pour ce faire, nous avons mis la valeur de la constante `k` à 0 afin d'empêcher la suppression de liens durant cette étape. La figure 5.2.3.1 montre l'état du réseau à la fin de cette étape.

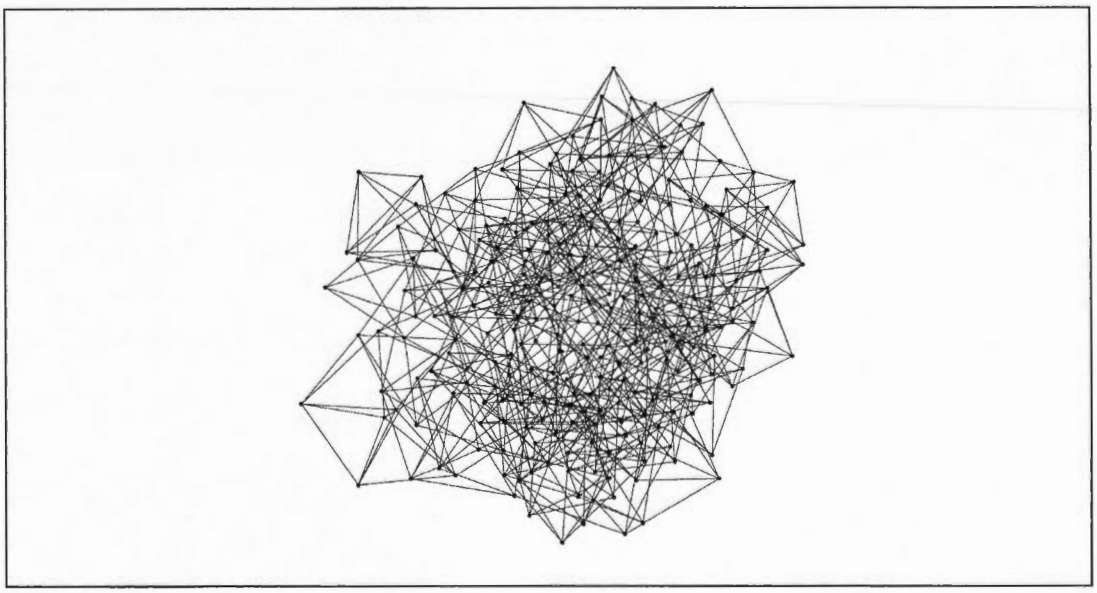


Figure 5.2.3.1 Réseau initial de la simulation du réseau d'amis à t = 0 (625 liens)

Une fois notre configuration initiale construite, nous avons amorcé la simulation de notre modèle en spécifiant des valeurs de paramètres très similaires à ceux utilisés par les auteurs du modèle. Certains paramètres n'étaient pas spécifiés explicitement dans l'article alors nous avons expérimenté avec différentes valeurs pour trouver celles qui semblaient donner de meilleurs résultats. Le tableau 5.2.3.1 montre la comparaison des valeurs des paramètres utilisées par les auteurs du modèle avec les valeurs des constantes utilisées pour exécuter la simulation dans NetSim. On remarque qu'à l'exception des valeurs non spécifiées par les auteurs, les valeurs utilisées pour la simulation avec NetSim sont toutes identiques sauf dans le cas de la force minimale limite d'une amitié (`limit_min_for_friendship`). En effet,

nous avons réalisé qu'en augmentant légèrement cette valeur, nous obtenions de meilleurs résultats quant à la démonstration de l'effet de transitivité et de la formation de petites communautés à l'intérieur du réseau qui sont des propriétés de plusieurs types de réseaux sociaux.

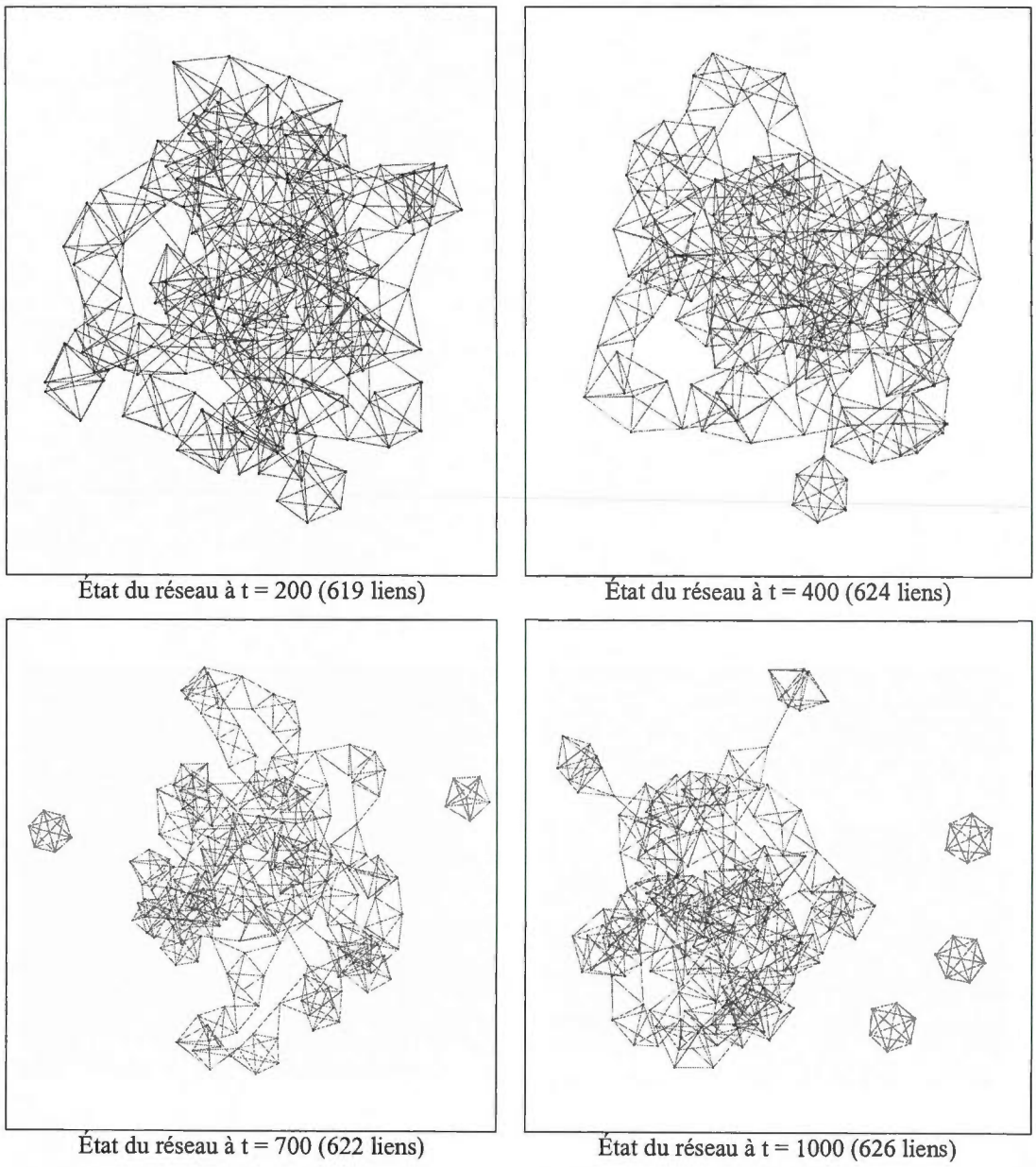
Tableau 5.2.3.1 Comparaison des valeurs des paramètres utilisées dans la simulation de NetSim avec celles utilisées dans la simulation des auteurs du modèle

Paramètres du modèle (des auteurs)	Constantes du modèle (Netsim)	Valeurs des paramètres (des auteurs)	Valeurs des constantes (NetSim)
k	k	0.01	0.01
β	b	5	5
z*	z_limite	5	5
p_0	p0	Non spécifiée exactement mais dite très petite	0.006
α	a	Non spécifié	0.5
Force minimale limite d'une amitié	Limit_min_for_friendship	0.3	0.5

La figure 5.2.3.2 montre l'évolution du réseau à différents moments dans le temps. Comme l'expliquent les auteurs du modèle, on peut observer la formation de communautés bien déterminées à l'intérieur du réseau. La plupart des groupements observés sont restés reliés entre eux, mais certains se sont déconnectés de la composante principale du graphe, au cours de la simulation. Cette propriété (qui n'était pas, à l'origine, une des caractéristiques du modèle) est évidemment observable dans le monde réel. L'émergence de ce phénomène s'explique par le fait que si, durant l'évolution du réseau, une région du graphe commence à montrer une plus grande densité de connexions que la moyenne alors, dans cette région, on observe conséquemment un nombre plus élevé de paires de sommets qui ont des liens communs. Ainsi, de nouvelles amitiés vont préférablement se former entre ces paires de

sommets en favorisant, à leur tour, la densité de connexions à l'intérieur de cette région et ainsi de suite. Il apparaît donc qu'une toute petite fluctuation initiale dans la densité du réseau peut provoquer l'évolution de communautés cohésives à l'intérieur du système.

De plus, on remarque que ces communautés semblent autosuffisantes. En effet, à l'intérieur de celles-ci, plusieurs paires d'individus ont nécessairement des amis communs ce qui produit la formation de plusieurs « triangles » d'amitié. Un triangle est une structure autosuffisante dans ce modèle. Chaque paire de sommets, dans un triangle, a un voisin commun qui est le troisième sommet du triangle. Donc, parce qu'ils ont un ami commun et qu'ainsi la rencontre entre chaque paire de sommets a plus de chances de se produire souvent, la force de connexion entre ces paires de sommets est continuellement renouvelée. Cela signifie qu'à l'intérieur d'une communauté, les liens perdurent plus longtemps, en moyenne, que ceux qui relient les communautés entre elles. Ce sont donc les amis communs qui forment et maintiennent les communautés.



État du réseau à t = 200 (619 liens)

État du réseau à t = 400 (624 liens)

État du réseau à t = 700 (622 liens)

État du réseau à t = 1000 (626 liens)

Figure 5.2.3.2 Évolution du réseau d'amis au cours de la simulation dans NetSim

Nous avons ensuite calculé les coefficients de transitivité à chacune des étapes de la simulation montrées à la figure 5.2.3.1 (réseau initial) et à la figure 5.2.3.2 (réseaux subséquents) à l'aide du logiciel d'analyse de réseaux sociaux « Ucinet » (Borgatti, Everett et Freemanm, 2002). Le coefficient de transitivité est la mesure de la densité des triangles dans le réseau. C'est le rapport entre le nombre de structures triangulaires (triangles fermés) et le

nombre de triplets (triangles ouverts et fermés) dans le réseau. Le tableau suivant montre l'évolution du coefficient de transitivité au cours de la simulation.

Tableau 5.2.3.2 Évolution du coefficient de transitivité au cours de la simulation du réseau d'amis

Étape	Coefficient de transitivité
Réseau initial à t = 0	0.309
Réseau à t = 200	0.440
Réseau à t = 400	0.473
Réseau à t = 700	0.516
Réseau à t = 1000	0.544

Les auteurs du modèle obtiennent un coefficient de transitivité égale à 0.45. Nous obtenons donc des résultats similaires si l'on considère notre simulation entre les temps t = 200 et t = 400. Comme ils le font remarquer, ce coefficient de transitivité est très élevé si on le compare à un graphe aléatoire de même dimension et possédant environ le même nombre de liens qui présente un coefficient de transitivité tournant autour de $z^* / N = 5 / 250 = 0.02$. N étant le nombre (constant) de nœuds dans le réseau et z^*, le nombre d'amis (liens) après lequel la possibilité de créer d'autres amitiés diminue radicalement. Ce modèle montre clairement la propriété de transitivité propre aux réseaux sociaux.

De plus, si l'on observe le tableau 5.2.3.2, on remarque que le coefficient de transitivité tend à augmenter avec le temps. Nous pourrions donc faire l'hypothèse que la formation de communautés se poursuit et s'intensifie avec le temps. On pourrait, éventuellement, observer l'évolution de la structure de ce réseau sur une plus grande période de temps pour évaluer cette hypothèse.

Il nous apparaît donc que notre simulation du modèle avec NetSim produit des résultats très similaires aux simulations effectuées par les auteurs du modèle.

5.3 Simulation d'un modèle de l'évolution du Web

5.3.1 Le modèle théorique

Comme modèle de l'évolution du Web, nous avons choisi de simuler le modèle proposé par (Albert et Barabási, 2000) qui est une extension du modèle indépendant de l'échelle précédemment introduit par (Barabási, Albert et Jeong, 1999). Ce modèle tient compte du phénomène de l'attachement préférentiel qui veut que les nœuds de degré élevé (nœuds populaires) tendent à attirer plus de liens que les nœuds de degré faible. Cette propriété a été observée dans certains réseaux tels que le WWW. De plus, contrairement au modèle d'amitié vu à la section précédente, ce modèle tient compte de la croissance constante du nombre de nœuds et de liens qu'on observe dans le WWW. En conséquence de cette croissance et du mécanisme d'attachement préférentiel, il a été montré (Albert, Jeong et Barabási, 1999) que la distribution des degrés du réseau engendré par l'évolution du Web semble suivre une loi de puissance. Nous tenterons donc de vérifier cette propriété sur le réseau engendré par notre simulation.

Ce modèle débute avec m_0 nœuds et à chaque pas de temps, les trois règles suivantes sont effectuées selon certaines probabilités.

1. **Ajout de nouveaux liens** : avec une probabilité p, ajouter $m < m_0$ liens au réseau. Pour ce faire, on choisit au hasard le nœud de départ de chaque nouveau lien. Ceci décrit, par exemple, un développeur Web qui décide d'ajouter un nouvel hyperlien à sa page Web. Le nœud d'arrivée du lien, par contre, est sélectionné avec la probabilité

$$\Pi(k_i) = \frac{k_i + 1}{\sum_j (k_j + 1)}$$ où k représente la connectivité du nœud (son degré).

Cette probabilité tient compte du fait que les nouveaux liens s'attachent préférentiellement à des nœuds populaires ayant déjà beaucoup de connexions. La figure 5.3.1.1 et le tableau 5.3.1.1 explicitent le calcul de cette probabilité.

2. **Reconnexion de liens** : avec une probabilité q, reconnecter m liens du réseau. Pour ce faire, on sélectionne au hasard un nœud i et un lien l_{ij} partant de i et allant à un autre noeud j. Ensuite, on supprime ce lien et on le remplace par un nouveau lien $l_{ij'}$ choisi selon la même probabilité $\prod$ qu'à la règle 1. Cette règle décrit, par exemple, la modification d'une URL sur une page Web vers une autre page Web.

3. **Ajout de nouveaux nœuds** : avec une probabilité $1 - p - q$, ajouter un nouveau nœud au réseau. Ce nouveau nœud possède m nouveaux liens qui sont connectés à m nœuds existants dans le graphe et choisis selon la même probabilité $\prod$ qu'à la règle 1. Cette règle représente l'ajout de nouvelles pages Web contenant un certain nombre d'hyperliens.

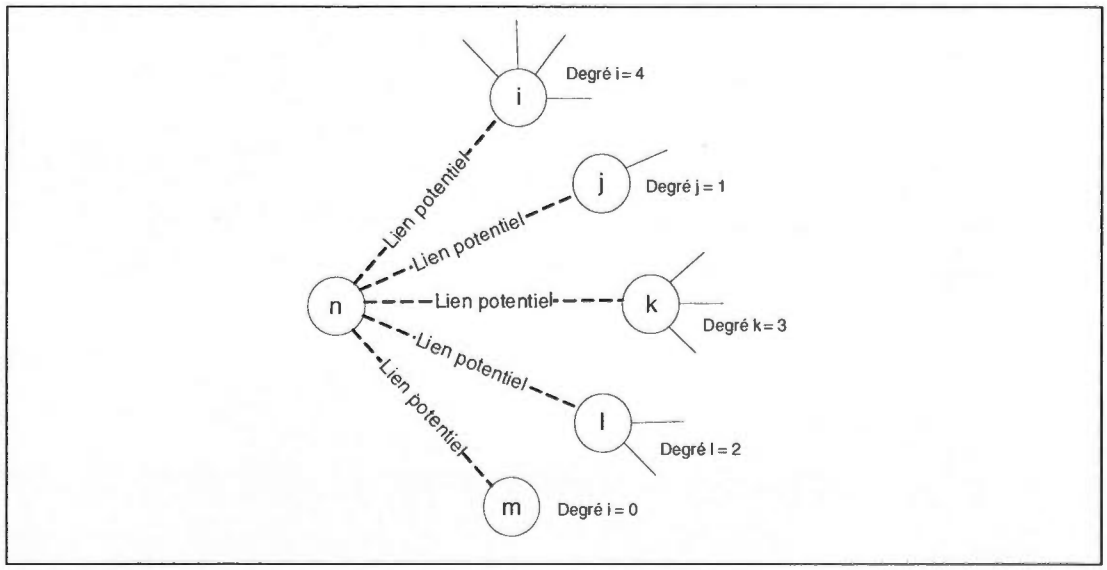


Figure 5.3.1.1 Attachement préférentiel dans le réseau du Web
Il s'agit de choisir un lien parmi tous les liens potentiels pouvant être ajouté au réseau et ayant comme premier nœud le nœud n. Le choix du second nœud se fait en tenant compte de sa connectivité (son degré). Plus un nœud a un degré élevé, plus il a de chances d'être choisi.

En se rapportant à la figure 5.3.1.1, le tableau 5.3.1.1 montre les calculs des probabilités $\prod$ de choisir les nœuds *i, j, k, l* ou *m* comme nœud d'arrivée du lien à ajouter. On remarque effectivement que cette probabilité tient compte de l'attachement préférentiel, car plus la connectivité d'un nœud est élevée, plus sa probabilité d'être choisi est élevée.

Tableau 5.3.1.1 Calcul de la probabilité $\prod$, pour l'ajout de nouveaux liens, qui tient compte de l'attachement préférentiel dans le modèle du Web

Nœud	Degré + 1	Calcul de $\prod$(degré) à 3 décimales près
i	5	5 / 15 = 0.333
j	2	2 / 15 = 0.133
k	4	4 / 15 = 0.267
l	3	3 / 15 = 0.200
m	1	1 / 15 = 0.067

Il est à noter que la probabilité $\prod(k_i)$ est proportionnelle à k_i+1 de telle sorte que les nœuds de degré zéro n'aient pas une probabilité nulle de recevoir de nouveaux liens.

Dans ce modèle, les probabilités p et q peuvent se situer dans les intervalles $0 \leq p < 1$ et $0 \leq q < 1-p$.

5.3.2 Le modèle traduit en langage NetSim

Les définitions du modèle

Les constantes

- La constante m représente le nombre de liens à ajouter ou à reconnecter selon la ou les règles exécutées à chaque pas de temps. Après avoir effectué certains tests de simulation, nous avons fixé cette valeur à 4.

```
DEFINE_CONST (m, 4);
```

- La constante p est la probabilité d'exécuter la règle 1. Selon nos différents tests de simulation, nous avons trouvé que la valeur 0.4 produisait de bons résultats.

```
DEFINE_CONST (p, 0.4);
```

- La constante q est la probabilité d'exécuter la règle 2. Après avoir testé plusieurs valeurs lors de simulations diverses, nous avons fixé cette valeur à 0.3.

```
DEFINE_CONST (q, 0.3);
```

Les attributs

- Bien qu'il n'y ait pas d'attributs définis par ce modèle, nous aurons besoin d'une variable témoin qui marque les nœuds traités. Nous allons donc définir cet attribut que nous initialisons à zéro.

```
DEFINE_ATTRIB (node, mark, 0);
```

Les fonctions

- Nous aurons besoin d'une fonction pour calculer la probabilité $\prod$. Pour ce faire, nous avons recours à la définition d'une méthode Java définie sur l'entité link que nous appellerons probToAddThisLink. Nous devons tout d'abord la déclarer avec la commande `DEFINE_JAVA_METHOD` comme suit:

```
DEFINE_JAVA_METHOD (link, probToAddThisLink);
```

Nous allons ensuite écrire le code java de cette méthode dans l'éditeur de code java du logiciel NetSim. Voici le code :

```
public Double probtoaddthislink(Link link) {

    Double value;              //Valeur à retourner
    double sumDegree = 0;      //somme des degrés de tous les liens
                               //potentiels

    // obtenir tous les liens potentiels qui ont comme noeud de départ
    // le nœud de départ du lien reçu en paramètre.

    ArrayList links = simController.
                      getPotentialNodeOutLinks(link.getStartNode());

    // Calculer la somme des degrés (+1) des noeuds d'arrivée de tous
    // les liens potentiels obtenus à l'étape précédente.

    for (int i = 0 ; i < links.size() ; i ++) {
        sumDegree = sumDegree +
          simController.getDegree(((Link)links.get(i)).getEndNode()) + 1;
    }

    // Calculer le rapport entre le degré (+1) du nœud d'arrivée du lien
    // passé en paramètre et la somme des degrés calculée à l'étape
    // précédente.

    value = new Double ((simController.getDegree(link.getEndNode()) + 1) /
        sumDegree);

    //Le résultat retourné est la probabilité de la création de ce lien
    //potentiel (passé en paramètre) qui tient compte de l'attachement
    //préférentiel.

    return value;
}
```

Les règles du modèle

Comme les règles du modèle s'effectuent selon une probabilité, il n'est pas certain qu'elles s'effectueront toutes dans un même pas de temps. Nous allons donc créer une phase différente pour chaque règle. La condition d'exécution de chaque phase dépend de la probabilité d'effectuer la règle concernée : la règle 1 s'exécute selon la probabilité p, la règle 2, selon la probabilité q et la règle 3 selon la probabilité `1 - p - q`.

Phase 1

La phase 1 contient l'action qui modélise la règle 1 concernant l'ajout de m liens au réseau. Pour fabriquer la condition d'exécution de la phase 1, nous appelons la méthode

`random_number()` qui retourne un nombre aléatoire entre 0 et 1 inclusivement. Si ce nombre est plus petit ou égal à la probabilité `p`, la condition est vraie et la phase 1 s'exécute, sinon, la phase 1 ne s'exécute pas.

L'action `ADD_LINKS` sélectionne tous les liens potentiels qui passent le test probabiliste donné par la méthode `probToAddThisLink` et parmi tous ces liens sélectionnés, elle en choisit `m` à ajouter au réseau.

```
phase_exec_condition (random_number() <= p);
ADD_LINKS (probToAddThisLink(), m);
```

Phase 2

La phase 2 contient toutes les actions qui modélisent la règle 2 concernant la reconnexion de `m` liens dans le réseau. La condition d'exécution de cette phase suit le même principe que celle de la phase 1.

L'action `CUT_LINKS` sélectionne `m` liens du réseau (ceux que l'on désire reconnecter) et, avant de les supprimer, marque le nœud de départ de chacun de ces liens en affectant la valeur 1 à leur attribut `mark`.

Ensuite, l'action `ADD_LINKS` sélectionne tous les liens potentiels dont le premier nœud a son attribut `mark` égal à 1 (nœud marqué à l'action précédente). Tous les liens ainsi choisis passent l'épreuve probabiliste donnée par la méthode `probToAddThisLink`. Tous les liens qui réussissent le test sont ajoutés au réseau et l'attribut `mark` de leur nœud de départ est remis à 0 pour qu'un autre lien, avec ce même nœud de départ, ne soit pas choisi de nouveau.

L'action `PICK_LINKS` a pour but de s'assurer que l'attribut `mark` de tous les nœuds du réseau est remis à 0 pour la prochaine phase ou le prochain pas de temps.

```
phase_exec_condition (random_number() < q);
```

```
CUT_LINKS (m, start_node().mark := 1);

ADD_LINKS (start_node().mark = 1, probToAddThisLink(),
           start_node().mark := 0);

PICK_LINKS(start_node().mark := 0);
```

Phase 3

La phase 3 contient toutes les actions qui modélisent la règle 3 concernant l'ajout, au réseau, d'un nœud avec `m` liens. La condition d'exécution de cette phase suit le même principe que pour la phase 1.

L'action `ADD_NODES` ajoute un nœud au réseau et marque ce nœud en affectant la valeur 1 à son attribut `mark`. On remarque ici que pour ajouter un seul nœud, il faut que la valeur soit légèrement plus grande que 1 car la valeur exacte de 1 est traitée, par convention, comme une probabilité (valeur entre 0 et 1 inclusivement) et non comme un nombre. Par convention, toute valeur plus grande que 1 est traitée comme un nombre entier et sa partie décimale, s'il y a lieu, sera tronquée. Comme nous l'avons déjà expliqué au chapitre 3, lorsqu'on traite une probabilité comme premier paramètre d'une action `ADD`, par convention, le système crée tout d'abord un nombre de nouveaux nœuds égal au nombre de nœuds déjà présents dans le réseau. Ensuite, chaque nouveau nœud doit passer le test probabiliste donné par ce paramètre probabiliste. Dans le cas d'une probabilité égale à 1, tous les nouveaux nœuds passeront l'épreuve. Ainsi, dans le cas de notre action ADD_NODES, cela aurait pour effet d'ajouter au réseau un nombre de nœuds égal au nombre de nœuds déjà présents dans le graphe, ce qui ne correspond pas à ce que nous voulons faire. Puisque la valeur 1.1 est traitée comme un nombre (par convention), sa partie décimale sera tronquée et un seul nœud sera ajouté au réseau (la valeur de l'attribut `mark` de ce nœud ayant préalablement été ajustée à 1). C'est exactement ce que nous voulons faire ici.

L'action `ADD_LINKS` débute en sélectionnant tous les liens potentiels dont le nœud de départ est marqué. Ensuite, les liens choisis passent le test probabiliste donné par la méthode

probToAddThisLink et parmi les gagnants, on sélectionne finalement m liens à ajouter au réseau.

L'action PICK_LINKS a pour but de s'assurer que l'attribut mark de tous les nœuds du réseau est remis à 0.

```
phase_exec_condition (random_number() < 1 - p - q);
ADD_NODES (1.1, mark := 1);
ADD_LINKS (start_node().mark = 1, probToAddThisLink(), m);
PICK_LINKS (start_node().mark := 0);
```

Maintenant que nous avons écrit le modèle du Web en langage NetSim, nous pouvons le simuler dans NetSim. La section suivante commente nos résultats obtenus par simulation.

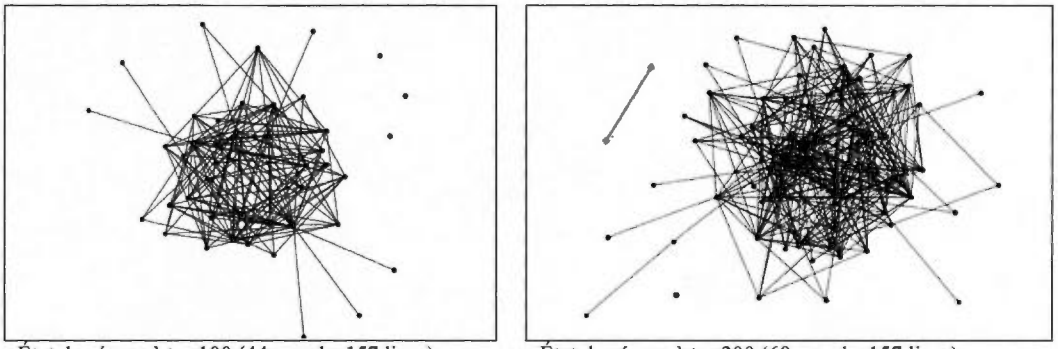
5.3.3 Résultats de simulation

Tout d'abord, nous avons construit notre réseau initial avec $m_0 = 6$ nœuds initiaux. Nous avons alors démarré la simulation et la figure 5.3.3.1 montre l'évolution de la structure du réseau à différents moments dans le temps. Dans l'image du réseau au temps t = 600, nous avons coloré certains nœuds de degré élevé en séparant la valeur des degrés selon des tranches de cinq. Étant donné que le degré maximum du réseau final est de 43, le code des couleurs des nœuds est le suivant :

Tranche des degrés	Nombre de noeuds	Couleur des noeuds
43 – 39	1	rouge
38 – 34	2	orange
33 – 29	2	jaune
28 – 24	6	vert
23 - 19	17	bleu

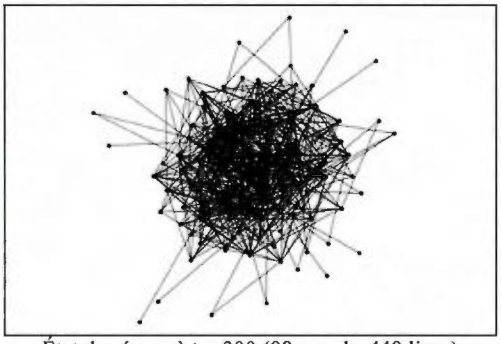
Nous remarquons déjà qu'il y a très peu de nœuds avec un degré très élevé se situant dans les trois premières tranches des degrés. Nous remarquons aussi que le nombre de nœuds semble

augmenter fortement à mesure que le degré diminue doucement. Ceci est une première indication de la possibilité d'une distribution exponentielle des degrés.

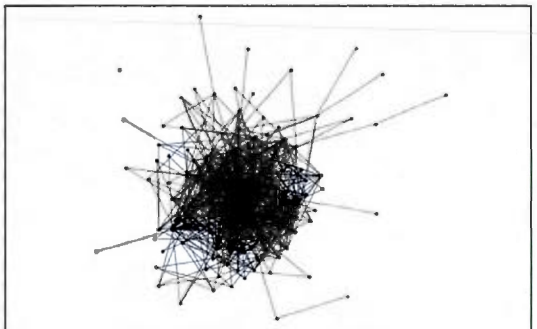


État du réseau à t = 100 (44 nœuds, 157 liens)

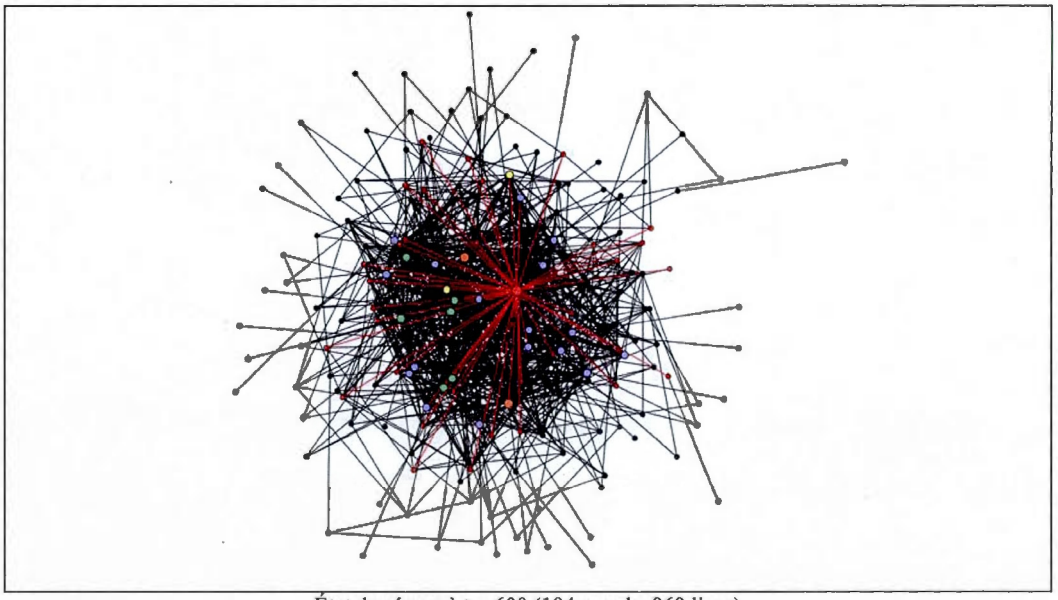
État du réseau à t = 200 (69 nœuds, 157 liens)



État du réseau à t = 300 (98 nœuds, 449 liens)



État du réseau à t = 400 (126 nœuds, 566 liens)



État du réseau à t = 600 (194 nœuds, 868 liens)

Figure 5.3.3.1 Évolution du réseau du Web au cours de la simulation

Cependant, nous voudrions voir si la distribution des degrés de notre réseau généré par simulation suit bien une loi de puissance de la forme $p(d) = d^{-\alpha}$ où $p(d)$ est la probabilité du degré d et α, un paramètre variable. Nous avons donc tracé graphiquement la distribution des degrés en omettant les degrés de valeur égale à 0 et les probabilités égales à zéro. La figure 5.3.3.2 montre ce graphique dans lequel l'axe des x correspond aux degrés des nœuds tandis que l'axe des y représente la probabilité de chaque degré dans le réseau. Nous constatons que les points du graphique semblent dessiner une courbe à l'allure exponentielle.

Si la courbe de la figure 5.3.3.2 suit une loi de puissance, nous devrions obtenir une droite en prenant le logarithme des valeurs en x et en y. La figure 5.3.3.3 illustre ce graphique. En utilisant une technique de régression linéaire, nous obtenons une pente de régression dont le coefficient de corrélation est égal à -0.879. Il semble que le modèle simulé avec NetSim produise effectivement une structure de réseau dont la distribution des degrés suit assez bien une loi de puissance.

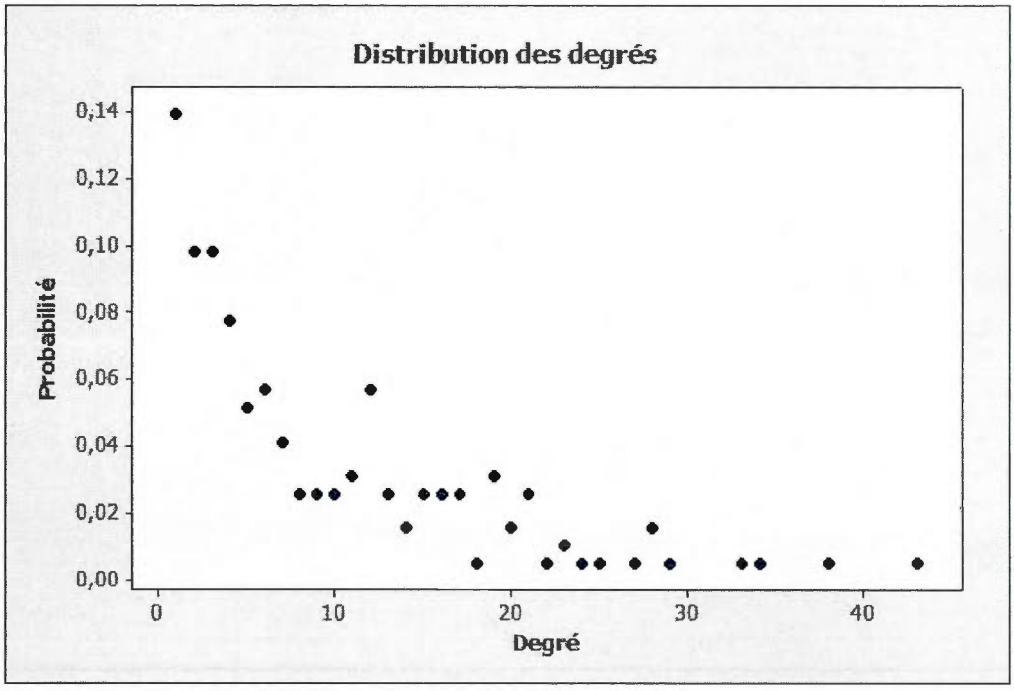


Figure 5.3.3.2 Distribution des degrés du réseau généré par simulation au temps t = 600
L'axe des x représente les degrés des nœuds du réseau et l'axe des y, les probabilités d'occurrence de ces degrés dans le réseau. Les points du graphique semblent dessiner une courbe de type exponentiel.

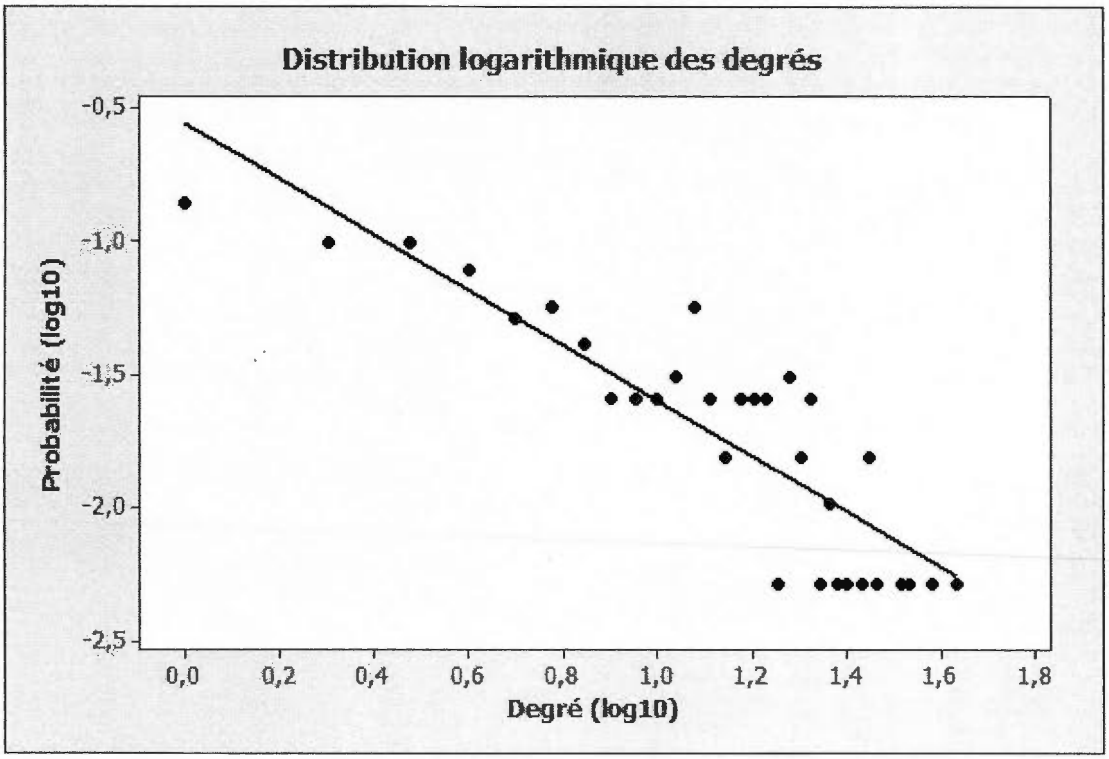


Figure 5.3.3.3 Distribution logarithmique des degrés du réseau généré par simulation au temps t = 600

Ce graphique illustre la distribution logarithmique du graphique de la figure 5.3.3.2. L'axe de x représente le logarithme des degrés des nœuds du réseau et l'axe des y, le logarithme des probabilités d'occurrence de ces degrés dans le réseau. Les points du graphique semblent dessiner une ligne droite qui indique que la distribution des degrés suit une loi de puissance. Le coefficient de corrélation de la pente de régression est assez significatif avec une valeur de -0.879. La valeur négative du coefficient de corrélation signifie que les données sont corrélées négativement : plus la valeur des degrés augmente, plus la valeur des probabilités diminue.

5.4 Conclusion

Dans ce chapitre, nous avons traduit en langage NetSim et simulé deux modèles typiques de la littérature. Bien que notre analyse des résultats obtenus reste assez sommaire, nous voulions d'abord montrer qu'une première analyse visuelle est souvent révélatrice des résultats que nous sommes susceptibles d'obtenir, par la suite, avec une analyse plus poussée.

Dans le cas du modèle d'amitié, nous avons obtenu des résultats très similaires à ceux obtenus par les auteurs du modèle. Le réseau généré par NetSim montrait effectivement la propriété de transitivité ainsi que la formation de petites communautés très cohésives au sein de la structure. Ces phénomènes sont aussi des phénomènes observés dans plusieurs réseaux sociaux réels.

En ce qui concerne le modèle de l'évolution de la structure du Web, nous avons obtenu, comme mentionné dans la littérature, un réseau dont la distribution des degrés semble effectivement suivre une loi de puissance.

CHAPITRE 6

Conclusion et perspective

6.1 Contributions

Dans ce mémoire, nous avons montré que l'étude des réseaux par lesquels circulent information et connaissances est pertinente dans notre société moderne et que l'investigation de ces réseaux par modélisation et simulation est, dans certains cas, une solution appropriée : elle permet de tester diverses hypothèses ou d'aider à en formuler d'autres quant aux phénomènes observés dans le monde réel.

Étant donné que l'échange d'informations se fait, le plus souvent, dans le cadre de réseaux sociaux, nous avons vu différents modèles de ce type de réseaux, tirés de la littérature et provenant de plusieurs domaines tels que la physique, les mathématique, la psychologie, la sociologie et l'économie. Parmi ces divers types de modèles, nous avons choisi de nous intéresser plus particulièrement à ceux qui tentent de comprendre la structure des interactions entre les acteurs du réseau plutôt qu'à ceux qui étudient la dynamique de ces interactions.

Cette « structure » peut dès lors être représentée sous forme de graphe et de ce fait, peut être examinée de façon quantitative par des méthodes d'analyse se basant sur la théorie des graphes. Nous en avons vu quelques-unes provenant de la sociologie structurale.

De plus, cette « structure », lorsque visualisée sous forme de graphe, peut déjà révéler plusieurs informations lorsqu'elle montre certains motifs spécifiques dans l'arrangement de ces nœuds et de ces liens.

Cependant, la mise en œuvre de simulations de modèles de réseaux d'information est souvent lente et laborieuse, car celles-ci sont souvent programmées à partir de zéro et ne répondent aux besoins que d'un seul modèle : l'implémentation est à recommencer lorsqu'il s'agit de simuler des modèles différents. C'est pourquoi nous avons proposé, dans ce mémoire, une plate-forme générique de modélisation de réseaux d'information et plus particulièrement, de réseaux sociaux.

En nous inspirant de modèles existants, nous avons tout d'abord élaboré un langage de modélisation qui est assez simple pour être accessible à des utilisateurs variés et assez concis pour être capable de spécifier un modèle en peu de commandes. Un modèle exprimé dans ce langage est facilement modifiable et permet de tester rapidement différentes règles ou différentes valeurs de paramètres de modèles divers.

Nous avons ensuite implémenté NetSim, un logiciel de simulation capable d'interpréter notre langage de modélisation et de simuler des modèles variés tout en visualisant leur évolution, au cours de la simulation, sous forme de graphe. Étant donné l'aspect structural de ces modèles, la visualisation est importante, car outre le fait qu'elle nous permet de déceler divers motifs ou régularités structurales significatives, elle permet aussi de nous apercevoir rapidement si le modèle simulé semble tendre ou non vers la structure recherchée.

Finalement, nous avons testé notre plate-forme de simulation sur différents modèles classiques de la littérature. Les résultats de simulations que nous avons obtenus sont plutôt convaincants, car ils sont significativement conformes aux résultats obtenus par les auteurs des différents modèles simulés.

La plupart des travaux techniques actuels ne tiennent pas suffisamment compte des aspects sociaux de la communication et restent ainsi en deçà de leur potentiel d'utilisation. Cet outil de modélisation nous donne la possibilité de modéliser finement des réseaux pertinents pour la circulation de l'information entre individus. Cela nous semble très utile pour le développement, par exemple, de nouvelles applications de communication électronique ou de travail collaboratif par ordinateur.

6.2 Travaux futurs

Pour la suite de ce projet, nous voudrions étendre les fonctionnalités de NetSim en y ajoutant un module d'analyse de réseaux. En effet, bien que la visualisation des réseaux soit révélatrice en soi, elle n'est pas suffisante pour bien comprendre et caractériser les structures générées par simulation. Nous pourrions commencer par implémenter les méthodes présentées au chapitre 2. Par exemple, pour une étude plus approfondie au niveau structural, il serait intéressant de pouvoir partitionner le réseau en sous-groupes cohésifs selon différentes méthodes comme les méthodes basées sur l'accessibilité et le diamètre, sur la réciprocité complète ou sur l'équivalence structurale et de pouvoir visualiser ces partitions à l'écran. Pour une analyse au niveau local, nous pourrions implémenter les méthodes de calcul des différents pointages de centralité et de prestige, etc. Il existe aussi des méthodes statistiques plus complexes d'analyse de réseau (Wassermann et Faust, 1994) que nous voudrions éventuellement implémenter.

De plus, le langage de modélisation NetSim est, pour le moment, à un état embryonnaire et nous voudrions le développer pour qu'il puisse modéliser une plus grande variété de réseaux d'information. Pour ce faire, nous voudrions d'abord améliorer les mécanismes de création de fonctions. Comme nous l'avons déjà mentionné, pour le moment, les fonctions se limitent à retourner un résultat numérique ou booléen. De plus, les fonctions ne peuvent pas recevoir de paramètres. Éventuellement, il serait intéressant de pouvoir créer des librairies de fonctions définies en langage NetSim. Chaque librairie pourrait fournir des méthodes spécialisées dans l'étude de phénomènes spécifiques ou de types particuliers de réseaux d'information. Elles pourraient être développées séparément par certains et se charger dans NetSim pour pouvoir être utilisées par d'autres. Le code NetSim pourrait ainsi être réutilisé.

Nous espérons que l'ébauche d'un tel langage de modélisation ouvrira la porte à d'autres propositions. Beaucoup de concepts de programmation sont déjà passés dans la pratique courante de l'informatique (langages numériques, langages de manipulation de listes, langages orientés objets...) et nous croyons qu'il est temps, maintenant, de se donner des outils pour modéliser les réseaux d'information.

ANNEXE A

Code généré dynamiquement des classes d'exécution de la simulation du modèle du réseau d'amis de Jin, Girvan et Newman

Classe SimulationContext

```
package org.netsim.simulation;
import org.netsim.network.*;
import org.netsim.utils.*;

public class SimulationContext extends Thread {
  private int phaseCounter = 0;
  private ModelPhases phase = null;
  private boolean suspend = true;
  private SimulationController simController =
      SimulationController.getController();

  public SimulationContext () {
    super();
    phase = null;
    phaseCounter = 0;
  }

  public void resetPhaseCounter () {
    phaseCounter = 0;
    phase = null;
  }

  public int getPhaseCounter () {
    return phaseCounter;
  }

  public synchronized void pauseThread () {
    suspend = true;
  }

  public synchronized void resumeThread () {
    suspend = false;
    notifyAll();
  }
```

```
    notifyAll();
  }
  public void run () {
    while (!(simController.getTimeStep() >= 1000.0)) {
      synchronized (this) {
        try {
          Thread.sleep((int) Math.random() * 1000000);
          while (suspend) {
            wait();
          }
        } catch (InterruptedException e) {
          System.out.println("Interrupted exception");
        }

        //phase 1 exec condition
        if (phase == null && true) {
            phaseCounter = 1;
            phase = new ModelPhase1();
        }

        if (phase != null) {
          phase.simule();
        }

        //update time step
        simController.incrementTimeStep();

        //refreshing the graph display
        simController.refreshDisplayedGraph();

      } //end synchro

    } //end while

    suspend = true;
    NetSimUtils.popInfoMsg(null, "Normal end of this simulation!");

  } // end method

} //end class
```

Classe ModelPhase1

```
package org.netsim.simulation;

import java.util.HashMap;
import java.util.ArrayList;
import org.netsim.network.*;
import org.netsim.utils.*;

public class ModelPhase1 extends ModelPhases {

  public void pickLinks0(SimulationController simController) {

    double [] numValues = {};
    ArrayList processList = initPickLinks();
    ArrayList chosenLinks = new ArrayList();
    HashMap linksToRewire = new HashMap();
    boolean fail = false;
    boolean stop = false;
    int i = 0;
    ILink link;
    boolean boolValue;
    double doubleValue;
    int counter = -1;
    int number;

    while (!stop && processList.size() > 0) {

      i = MathUtils.randomNumber(0, processList.size() - 1);
      link = (ILink)processList.get(i);
      //ASSIGNMENT
      if (!fail) {
        simController.setAttribute(link, "time_since_last_meeting",
            simController.getAttribute(link,
            "time_since_last_meeting") + 1.0);
      }
      //ASSIGNMENT
      if (!fail) {
        simController.setAttribute(link, "strength",
            ( Math.pow(2.718, - 0.01 *
            simController.getAttribute(link,
            "time_since_last_meeting")) ));
      }
      fail = false;
      processList.remove(i);
      counter = -1;

    } //end while
  } //end method
```

```
public void pickLinks1(SimulationController simController) {

  double [] numValues = {};
    ArrayList processList = initPickLinks();
    ArrayList chosenLinks = new ArrayList();
    HashMap linksToRewire = new HashMap();
    boolean fail = false;
    boolean stop = false;
    int i = 0;
    ILink link;
    boolean boolValue;
    double doubleValue;
    int counter = -1;
    int number;

    while (!stop && processList.size() > 0) {
      i = MathUtils.randomNumber(0, processList.size() - 1);
      link = (ILink)processList.get(i);

      doubleValue = ( ( 1.0 - ( 1.0 - 0.0060 ) *
          Math.pow(2.718, - 0.5 *
          simController.getNumOfCommonNeighbours(link)) ) *
          ( 1.0 / ( Math.pow(2.718, 5.0 *
          ( simController.getDegree(simController.
          getStartNode(link)) - 5.0 )) + 1.0 ) ) * ( 1.0 / (
          Math.pow(2.718, 5.0 * (
          simController.getDegree(simController.
          getEndNode(link)) - 5.0 )) + 1.0 ) ) );

       //PROBABILITY
       if (!fail) {
         if (!MathUtils.isARandomWinningNumber(doubleValue)) {
           fail = true;
         }
       }
       //ASSIGNMENT
       if (!fail) {
         simController.setAttribute(link,
             "time_since_last_meeting", 0.0);
       }
       //ASSIGNMENT
       if (!fail) {
         simController.setAttribute(link, "strength", 1.0);
       }
       fail = false;
       processList.remove(i);
       counter = -1;

     } //end while
} //end method
```

```
public void cutLinks2(SimulationController simController) {

  double [] numValues = {};
  ArrayList processList = initCutLinks();
  ArrayList chosenLinks = new ArrayList();
  HashMap linksToRewire = new HashMap();
  boolean fail = false;
  boolean stop = false;
  int i = 0;
  ILink link;
  boolean boolValue;
  double doubleValue;
  int counter = -1;
  int number;

  while (!stop && processList.size() > 0) {
    i = MathUtils.randomNumber(0, processList.size() - 1);
    link = (ILink)processList.get(i);

    //BOOLEAN
    if (!fail) {
      boolValue = simController.getAttribute(link, "strength")
          <= 0.5;
      fail = !boolValue;
    }
    //ENDING
    fail = false;
    processList.remove(i);
    counter = -1;
  }//end while
} //end method
```

```
public void addLinks3(SimulationController simController) {
  double [] numValues = {};
  ArrayList processList = initAddLinks();
  ArrayList chosenLinks = new ArrayList();
  HashMap linksToRewire = new HashMap();
  boolean fail = false;
  boolean stop = false;
  int i = 0;
  ILink link;
  boolean boolValue;
  double doubleValue;
  int counter = -1;
  int number;

  while (!stop && processList.size() > 0) {
    i = MathUtils.randomNumber(0, processList.size() - 1);
    link = (ILink)processList.get(i);

    //add the link to the network
    simController.addLink(link);

    doubleValue = ( ( 1.0 - ( 1.0 - 0.0060 ) *
        Math.pow(2.718, - 0.5 *
        simController.getNumOfCommonNeighbours(link)) ) *
        ( 1.0 / ( Math.pow(2.718, 5.0 *
        ( simController.getDegree(simController.
        getStartNode(link)) - 5.0 )) + 1.0 ) ) * ( 1.0 / (
        Math.pow(2.718, 5.0 *
        ( simController.getDegree(simController.
        getEndNode(link)) - 5.0 )) + 1.0 ) ) );

    //PROBABILITY
    if (!fail) {
      if (!MathUtils.isARandomWinningNumber(doubleValue)) {
        fail = true;
      }

    }
    //ENDING
    if (fail) {
      simController.cutLink(link);
    }
    fail = false;
    processList.remove(i);
    counter = -1;
  } //end while

} //end method
```

```
public void simule() {

    SimulationController simController =
        SimulationController.getController();

    pickLinks0(simController);

    pickLinks1(simController);

    cutLinks2(simController);

    addLinks3(simController);

  }

}  //end class
```

ANNEXE B

Interface graphique de NetSim

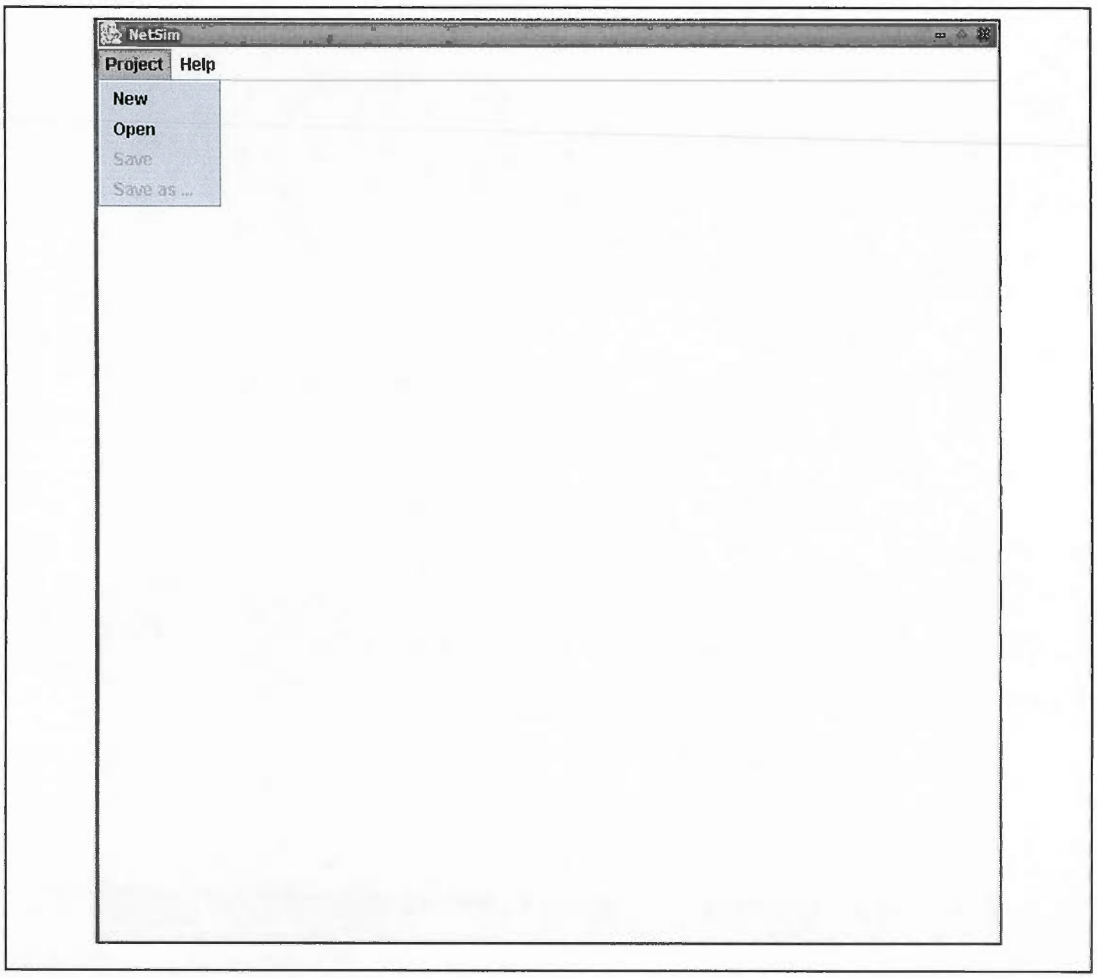


Figure B.1 Création d'un nouveau projet
Avant de créer une simulation dans NetSim, il faut tout d'abord créer un projet. Le même projet pourra, par la suite, contenir une ou plusieurs simulations.

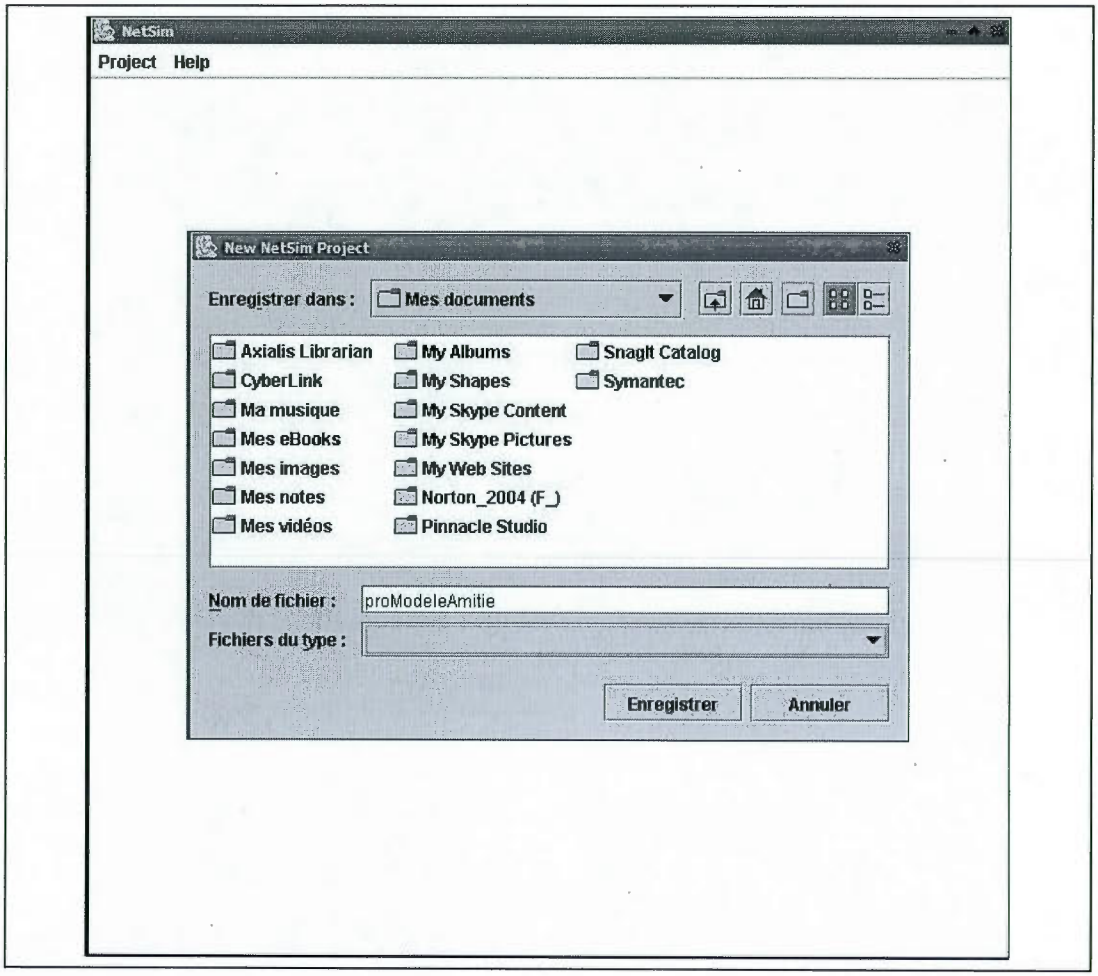


Figure B.2 Enregistrement du nouveau projet sur le disque
Lors de la création d'un nouveau projet, NetSim demande de lui fournir un nom pour ce projet et de lui donner l'emplacement où l'enregistrer.

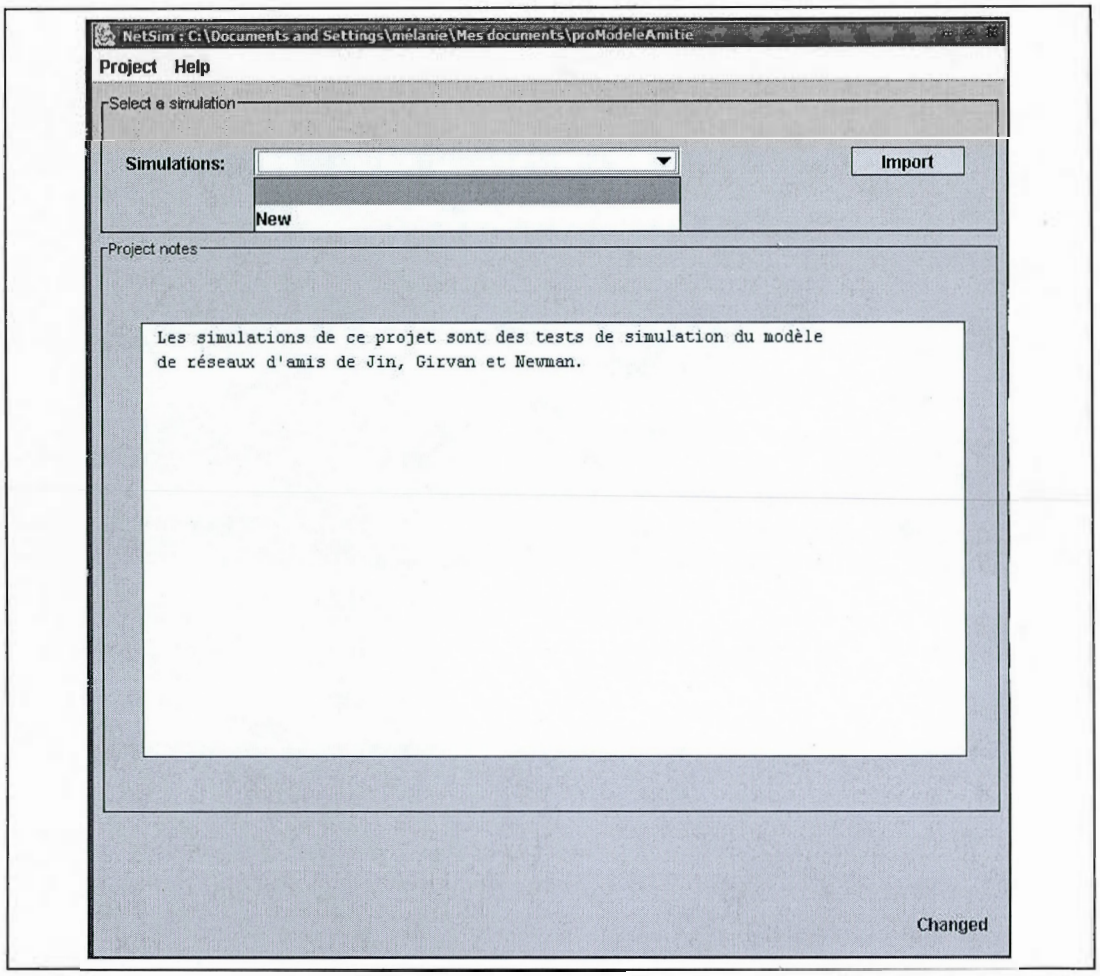


Figure B.3 Écriture de notes de projet et création d'une nouvelle simulation
À la création d'un projet, un espace est réservé pour l'écriture de notes à propos de ce projet. Ensuite, pour créer une nouvelle simulation, il faut sélectionner l'option « New » dans la liste des simulations. S'il y avait déjà des simulations d'enregistrées dans ce projet, elles apparaîtraient dans la liste des simulations et il serait possible de les sélectionner pour les ouvrir.

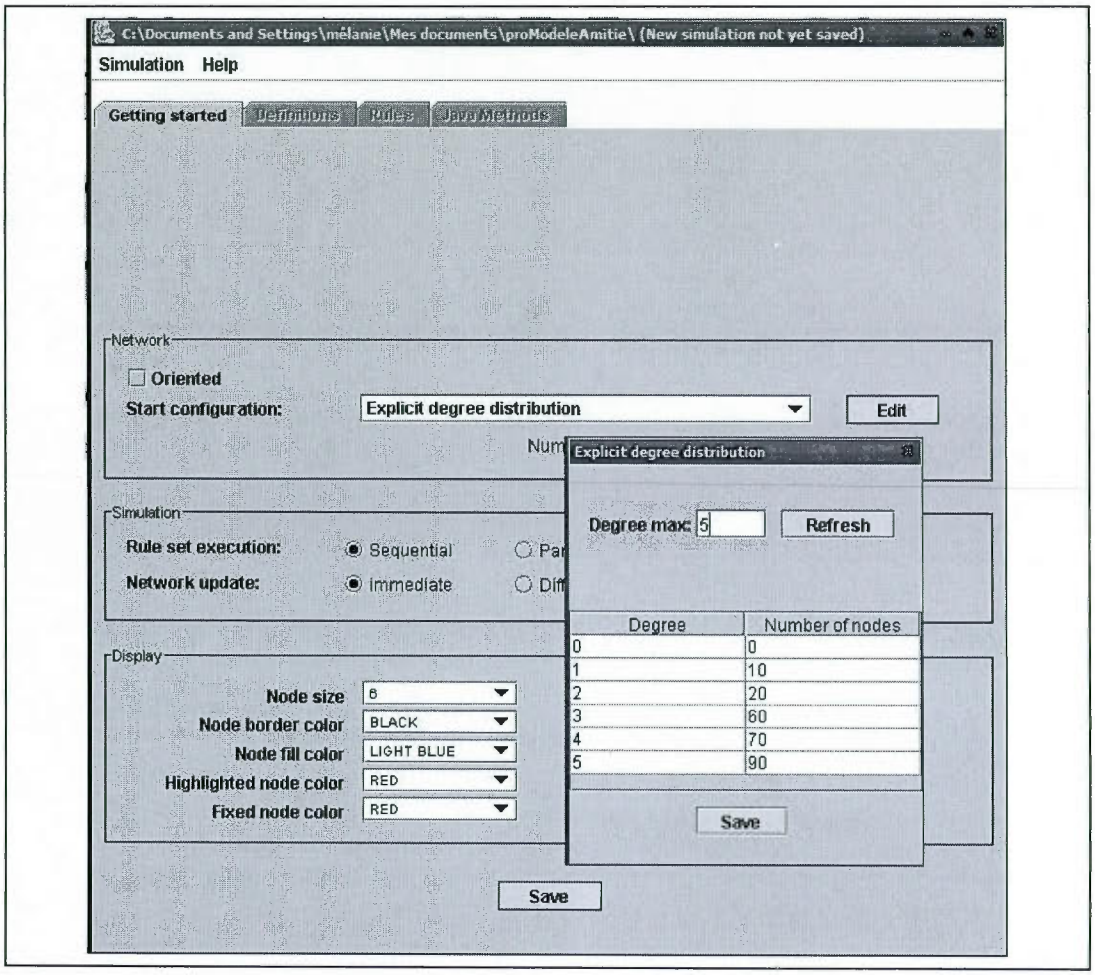


Figure B.4 Choix d'une configuration initiale dans les paramètres généraux de la simulation
On peut choisir une configuration pour le réseau initial de la simulation en choisissant une configuration dans la liste des « start configuration ». Dans cet exemple, on a sélectionné la configuration « Explicit Degree Distribution » et une autre fenêtre s'est ouverte, dans laquelle on doit entrer les valeurs des paramètres pour cette configuration. Dans ce cas, il faut tout d'abord donner le degré maximum de notre réseau et ensuite fournir le nombre de nœuds pour chaque degré allant de 0 à degré maximum.

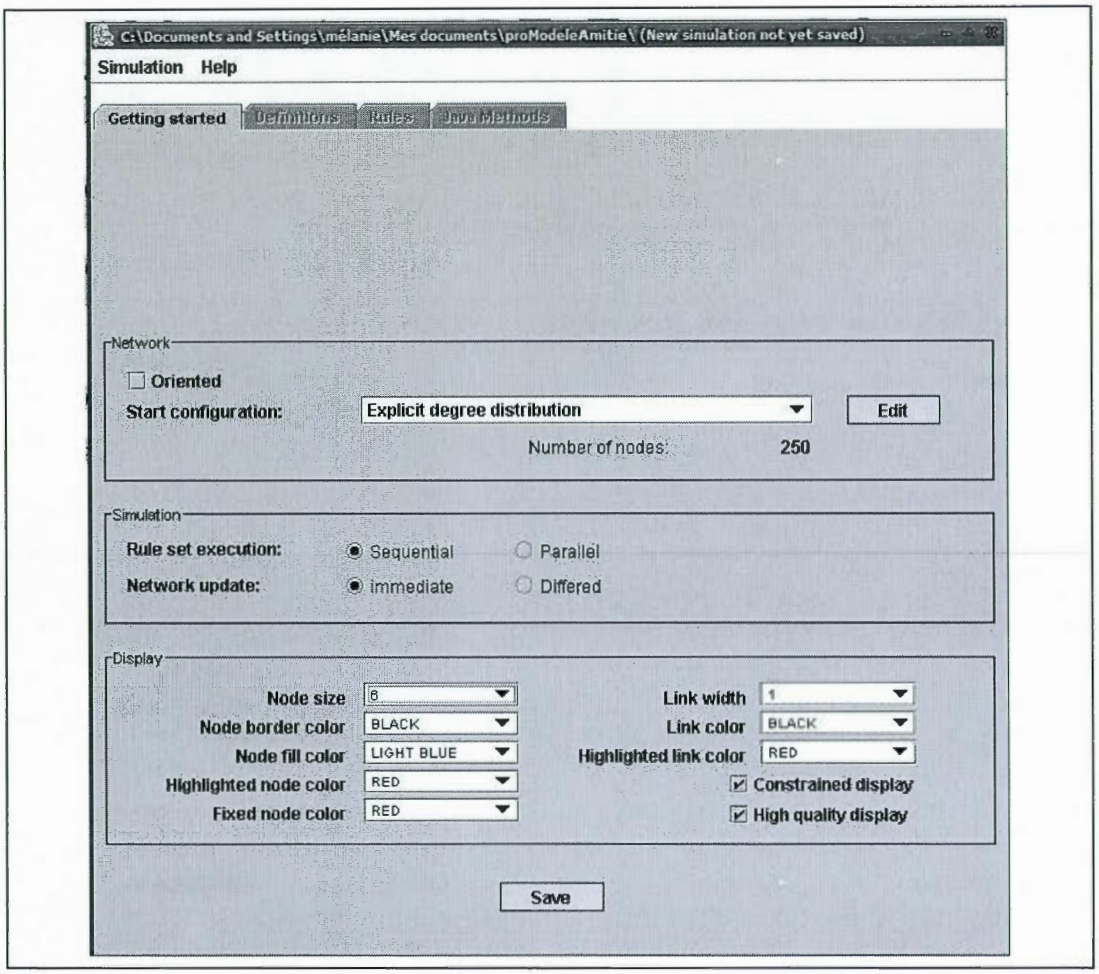


Figure B.5 Choix des autres paramètres généraux de la simulation
Dans l'onglet des paramètres généraux (getting started) de la simulation, on doit mentionner si le graphe est orienté ou non, le mode d'exécution des phases, le mode de mise à jour du réseau (pas encore implanté) les différents attributs visuels des nœuds et des liens, etc. Lorsqu'on appuie sur le bouton « Save », l'onglet suivant (Definitions) est activé et l'on peut écrire les définitions du modèle (voir figure suivante).

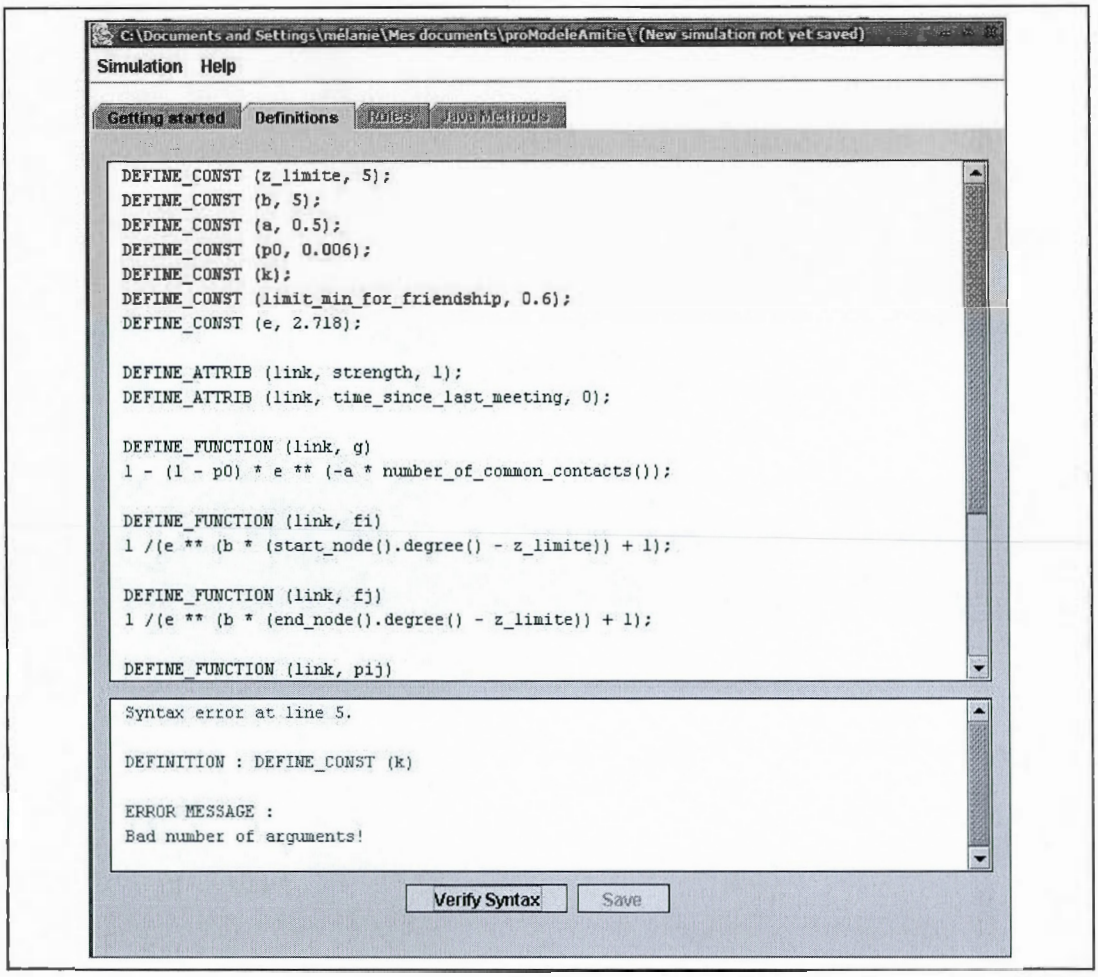


Figure B.6 Écriture des définitions du modèle et vérification syntaxique (avec erreurs)
C'est dans l'onglet « Définitions » que l'on écrit les définitions du modèle. Lorsqu'on clique sur le bouton « Verify Syntax », si nos définitions contiennent des erreurs de syntaxe, un message d'erreur (en rouge) est affiché dans l'aire de texte du bas.

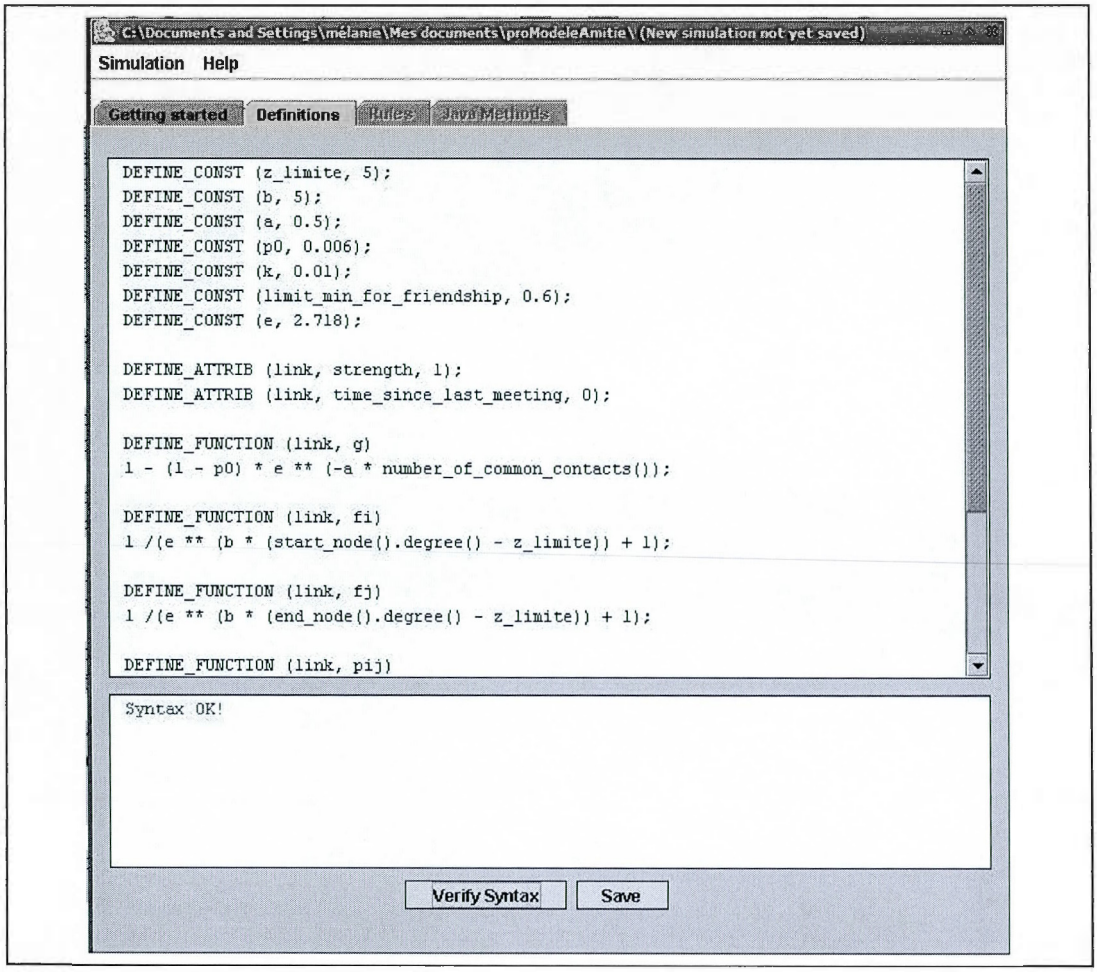


Figure B.7 Vérification syntaxique des définitions du modèle (sans erreurs)
Lorsque la syntaxe de nos définitions est correcte, le bouton « Save » devient actif et si l'on clique sur ce bouton, l'onglet suivant (Rules), dans lequel on peut écrire les règles du modèle, devient actif.

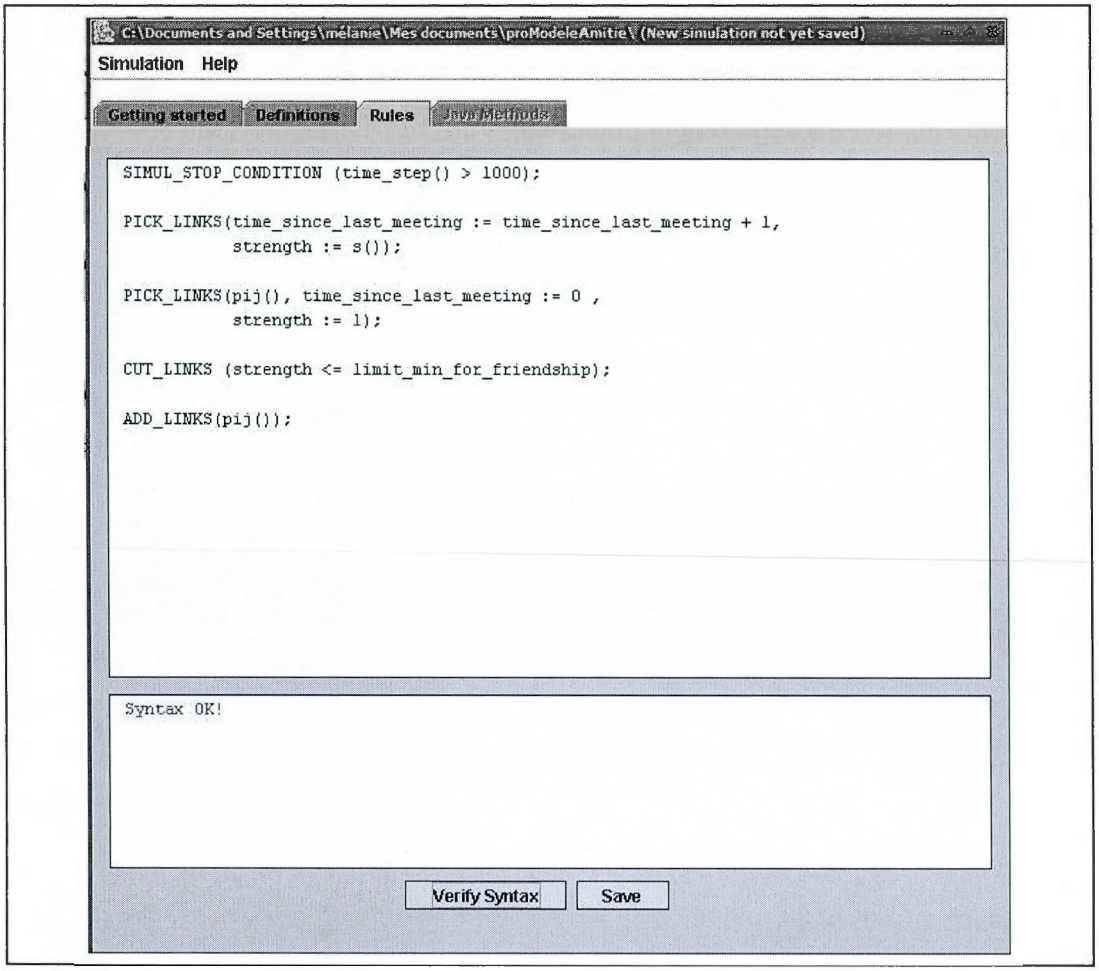


Figure B.8 Écriture des règles du modèles et vérification syntaxique (sans erreurs)
Lorsque la syntaxe de nos règles est correcte, le bouton « Save » s'active. Lorsque l'on clique sur ce bouton, si nous avons défini une méthode Java dans les définitions, l'onglet suivant (Java Methods), dans lequel on peut écrire notre méthode en Java, deviendra actif. S'il n'y a pas de méthode Java, le bouton « Start » apparaît (voir figure suivante).

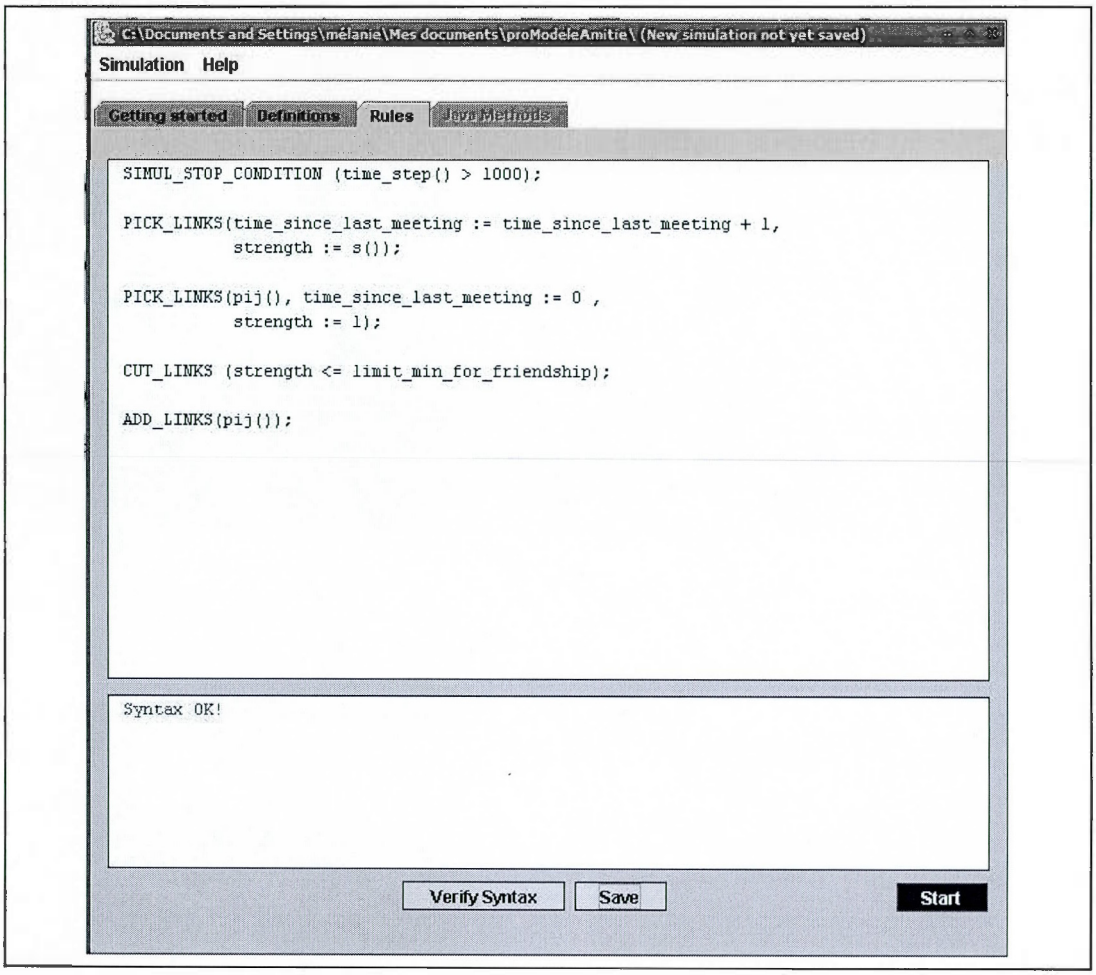


Figure B.9 Initialisation de la simulation (bouton « start »)
Le bouton « Start » sert à initialiser la simulation.

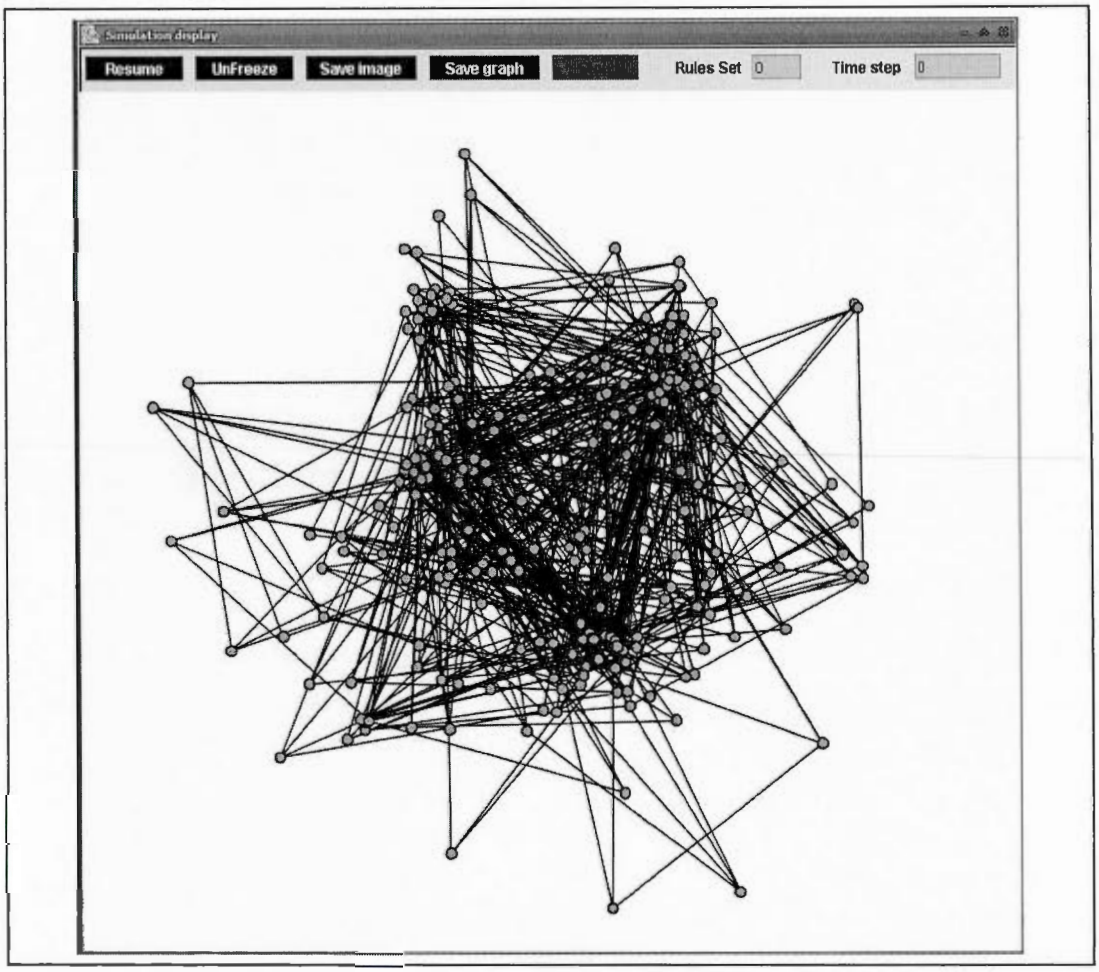


Figure B.10 Visualisation de la configuration initiale du réseau
À l'initialisation de la simulation, la configuration initiale du réseau, s'il y a lieu, est affichée à l'écran. Pour démarrer la simulation, il suffit de cliquer sur le bouton « Resume ». Pour arrêter la simulation, il faut cliquer sur le bouton « Stop ». En tout temps au cours de la simulation, on peut sauvegarder une image du réseau en cliquant sur le bouton « Save Image » ou sauvegarder la configuration du réseau sous le format NetSimXML et cliquant sur le bouton « Save graph ».

RÉFÉRENCES

Albert, R.H. et Barabási A.-L. (2000), Topology of evolving networks : local events and universality, *Physical Review Letters*, vol. 85, 5234.

Albert R., Jeong H. et Barabási A.-L. (1999), Diameter of the World-Wide Web, *Nature*, vol. 401, 130-131.

Amaral, L. A. N., Scala A., Barthélémy M. et Stanley H. E. (2000), Classes of small-world networks, *Proceedings of National Academy of Sciences USA*, vol. 97, 11149-11152.

Amblard F. (2002), Which ties to choose? A survey of social networks models for agent-based social simulations, *Proceedings of the 2002 SCS International Conference On Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, Portugal, Lisbon, avril, 253-258.

Auer K. et Norris T. (2001), "Arrieros Alife", a multi-agent approach simulating the evolution of a social system : modeling the emergence of social networks with Ascape, *Journal of Artificial Societies and Social Simulation*, vol. 4, no 1.
URL : http://www.soc.surrey.ac.uk/JASSS/4/1/6.html.

Baker G. L. et Gollub J. P. (1990), *Chaotic dynamics : an introduction*, Cambridge University Press, Cambridge.

Bala V. et Goyal S. (1999), A non-cooperative model of network formation, *Econometrica*, vol. 68, 1181-1230.

Banks D. L. et Carley K. M. (1994), Metric inference for social networks, *Journal of Classification*, vol. 11, 121-149.

Barabási A.-L., Albert R. et Jeong H. (1999), Mean-field theory for scale-free random networks, *Physica A*, vol. 272, 173-187.

Borgatti S. P., Everett M. G. et Freeman L. C. (2002), *UCINET 6 for Windows, Software for Social Network Analysis*, Harvard, Analytic Technologies.

Borgatti S. P., Everett M. G. et Freeman L. C. (2002), *UCINET 6 for Windows, Software for Social Network Analysis*, Harvard, Analytic Technologies.

Blau P. M. (1967), *Exchange and power in social life*, J. Wiley, New York.

Brin S. et Page L. (1998), The anatomy of a large-scale hypertextual Web search engine, *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australie, avril, 107-117.

Broder A., Kumar R., Maghoul F., Raghavan P., Rajagopalan S., Stata R., Tomkins A. et Wiener J. (2000) *Computer Networks*, vol. 33, 309-320.

Davis J. A. (1967), Clustering and structural balance in graphs, *Human Relations*, vol. 20, 181-187.

Donald M. (1993), Cognitive evolution and the definition of human nature, *Philosophy of Science lecture Series*.

Dorogovtsev S. N. et Mendes J. F. F. (1999) Exactly solvable analogy of small-world networks, submitted to *Physical Review Letters*.

Dorogovtsev S. N. et Mendes J. F. F. (2000), Evolution of reference networks with aging, *Physical Review E*, vol. 62, 1842.

Ellis C. A., Gibbs S. J. et Rein G. L. (1991), Groupware : some issues and experiences, *Communications of the ACM*, vol. 34, no 1, 38-58.

Epstein J. M. et Axtell R. (1996), *Growing artificial societies, social sciences from the bottom up*, MIT Press, Cambridge.

Erdös P. et Rényi A. (1960), On the evolution of random graphs, *Publications of the Mathematics Institute of Hungarian Academy of Science*, vol. 5, 17-61.

Flache A. et Hegselmann R. (2001), Do irregular grids make a difference? Relaxing the spatial regularity assumption in cellular models of social dynamics, *Journal of Artificial Societies and Social Simulation*, vol. 4, no 4.
URL : http://www.soc.surrey.ac.uk/JASSS/4/4/6.html.

Flache A. et Macy M. W. (1996), The weakness of strong ties : collective action failure in a highly cohesive group, *Journal of Mathematical Sociology,* vol. 21, 3-28.

Guare J. (1990), *Six Degrees of Separation : a play*, Vintage, New York.

Heider F. (1958), *The Psychology of Interpersonal Relations*, J. Wiley, New York.

Holland P. W. et Leinhardt S. (1981), An exponential family of probability distributions for directed graphs, *Journal of the American Statistical Association*, vol. 76, 33-65.

Hyatt A., Contractor N. et Jones P. M. (1997), Computational organizational network modeling : strategies and an example, *Computational and Mathematical Organizational Theory*, vol. 4, 285-300.

Jackson M. O. (2001), The stability and efficiency of economic and social networks, in *Models of the Formation of Networks and Groups*, B. Dutta and M. O. Jackson (eds.), Springer-Verlag, Heidelberg.

Jin E. M., Girvan M., Newman M. E. J. (2001), *The structure of growing social networks*, Working Papers 01-06-032, Santa Fe Institute.

Jordan K., Hauser J. et Foster S. (2003), The augmented social network: Building identity and trust into the next-generation Internet, in *First Monday*, vol. 8, no 8. URL: http://firstmonday.org/issues/issue8_8/jordan/index.html.

Kautz H., Selman B. et Shah M. (1997), Referral Web: combining social networks and collaborative filtering, *Communications of the ACM* 40 (3).

Kleinberg J. (1999), *The small-world phenomenon : an algorithmic perspective*, Cornell University, Computer Science Department, Technical Report 99-1776. http://www.cs.cornell.edu/home/kleinber/swn.ps.

Kirsch S., Gnasa M. et Cremers A. (2006), Beyond the Web: retrieval in social information spaces, *Proc. 28th European Conference on Information Retrieval.*

Korte C. et Milgram S. (1970), Acquaintance linking between white and negro populations : application of the small world problem, *Journal of Personality and Social Psychology*, vol. 15, 101–118.

Lamy C. (2004), Les logiciels sociaux : des outils qui placent l'individu au premier plan, *Bulletin SISTech.* URL : http://www.infometre.cefrio.qc.ca/loupe/sistech/0204.asp

Lazega E. (1998), *Réseaux sociaux et structures relationnelles*, Paris, Que sais-je? No 3399, PUF.

Memmi D. et Nérot O. (2003), Building virtual communities for information retrieval, in *Groupware: Design, Implementation and Use*, Favela et Decouchant (eds), Springer, Berlin.

Milgram S. (1967), The small world problem, *Psychology Today*, vol. 2, 60–67.

Mosler H.-J. et Brucks W. (2001), Social influence among agents, in *Cooperative Agents, Application in the Social Sciences*, N. J. Saam et B. Schmidt (eds), Kluwer, Amsterdam, 125-147.

Mutton P. (2004), Inferring and visualizing social networks on Internet relay chat, *Proc Information Visualization*, IEEE Computer Society, London, UK.

Myerson R. B. (1977), Graphs and cooperation in games, *Mathematics of Operations Research*, vol. 2, 225-229.

Newcomb T. M. (1968), Interpersonal balance, in *Theories of Cognitive Consistency: A Sourcebook*, R. P. Abelson et al. (eds.), Rand-McNally, Chicago, 28-51.

Newman M. E. J. (1999), *Small worlds : the structure of social networks*, Working Papers 99-12-080, Santa Fe Institute.

Newman M. E. J. (2001), Clustering and preferential attachment in growing networks, *Physical Review E*.

Nonaka I. (1991), The knowledge-creating company, *Harvard business review*, November-December.

Nowak A. et Vallacher R. R. (1998), *Dynamical social psychology*, Guildford Press, New York.

Peyton Young H. (1998), *Individual strategy and social structure : an evolutionary theory of institutions*, Princeton University Press, Princeton.

Poulin I. (2004), Combiner technologies de l'information et innovation d'affaires pour augmenter la productivité, *Bulletin SISTech*, 13 février 2004.
URL : http://www.infometre.cefrio.qc.ca/loupe/sistech/0204.asp#bas

Slikker M. et Van Den Nouweland A. (2001), A one-stage model of link formation and payoff division, *Games and Economic Behavior*, vol. 34, 153-175.

Snijders T. A. B. (1996), Stochastic actor-oriented models for network change, *Journal of Mathematical Sociology*, vol 21, 57-76.

Stokman F. N. et Van Oosten R. (1994), The exchange of voting positions : an object-oriented model for policy networks, in *European Community Decision Making. Models, applications, and comparisons*, B. Bueno de Meqsuita et F. N. Stokman (eds.), Yale University Press, New Haven, 105-127.

Stokman F. N. et Zeggelink E. (1996), Is politics power or policy oriented? A comparative analysis of dynamics access models in policy networks, *Journal of Mathematical Sociology*, vol. 21, 77-111.

Van Duijn M. et Snijders T. A. B. (1995), *The P2 Model*, Internal publication, VSM, University of Groningen.

Viegas F., Boyd D., Nguyen D., Potter J. et Donath J. (2004), Digital artifacts for remembering and storytelling : posthistory and social network fragments, *Proceedings of the 37th Hawaii International Conference on System Sciences*, janvier, 40109a.

Wasserman S. et Pattison P. (1996), Logistic models and logistic regressions for social networks : An introduction to Markov graphs and p*, *Psychometrika*, vol. 61, 401-425.

Wassermann S. et Faust K. (1994), *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge.

Watts D. et Strogatz S. H. (1998), Collective Dynamics of small world networks, *Nature*, vol. 393, 440.

Watts D. J. (1999) *Small Worlds*, Princeton University Press, Princeton, NJ.

Wolfram S. (1986), *Theory and Applications of Cellular Automata*, World Scientist, Singapore.

Zeggelink E. (1993), *Strangers into Friends, The evolution of friendship networks using an individual oriented modeling approach*, Thesis Publishers, Amsterdam.