Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet de programmation et génie logiciel

2018-2019

Naissance d'un langage

Tuteur académique
Christophe Lenté

Étudiants
Charles MECHERIKI (DI4)
Yongda LIN (DI4)

10 janvier 2019

Liste des intervenants

Nom	Email	Qualité
Charles MECHERIKI	charles.mecheriki@etu.univ-tours.fr	Étudiant DI4
Yongda LIN	yongda.lin@etu.univ-tours.fr	Étudiant DI4
Christophe Lenté	christophe.lente@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Charles MECHERIKI et Yongda LIN susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Christophe Lenté susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Charles MECHERIKI et Yongda LIN, *Naissance d'un langage*, Projet de programmation et génie logiciel, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
    author={MECHERIKI, Charles and LIN, Yongda},
    title={Naissance d'un langage},
    type={Projet de programmation et génie logiciel},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2018-2019}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.2.5 (2017/05/02)

Table des matières

Table des figures

Table des figures

1 Introduction

1.1 Context

La langue a toujours été le mécanisme de transmission de l'information le plus développé chez l'Homme.
Contrairement au langage binaire, aujourd'hui parfaitement connu et compris, le développement des langues humaines est un phénomène archaïque très complexe, suscitant un grand intérêt dans le monde scientifique.

En effet, que ce soit dans le domaine informatique avec l'essor de l'intelligence artificielle, en biologie (génétique) ou encore dans les sciences sociales (histoire, anthropologie, linguistique), la langue représente un outil prodigieux et mystérieux.

Chez l'Homme, l'évolution de la langue fut un processus extrêmement long, qui a commencé il y a plus de deux millions d'années.
La communauté scientifique a peu de certitudes sur les langues les plus anciennes, si ce n'est qu'elles seraient divisées en environ 300 familles réparties sur le globe, et que les langues actuelles en seraient les descendantes.
Cette évolution repose sur un modèle complexe, formé à la fois de phénomènes biologiques et de critères culturels, qui est encore aujourd'hui indéterminé.

Alors qu'il est impossible de reconstituer un tel processus étalé sur des millénaires, il est possible par **simulation informatique** de reproduire artificiellement un phénomène s'en approchant, dans le but d'établir des hypothèses sur son **modèle d'évolution**.

1.2 Objectifs

Ainsi, notre projet, intitulé "Naissance d'un langage", avait pour objectif la mise en place d'une simulation semblable.
Étant donné qu'un tel projet représente un défi colossal, nous avons bien évidemment dû considéré une version simplifiée.

Plus précisément, la formulation de l'objectif de notre projet s'apparentait à :

Soit un ensemble de villages, possédant chacun un vocabulaire limité et pouvant communiquer entre eux. À tour de rôle, les villages tentent d'émettre un mot de leur vocabulaire à leurs voisins, qui, s'ils le reçoivent, peuvent ou non adopter son usage. On s'intéresse à l'évolution du vocabulaire des différents villages au cours du temps.

2 Spécifications du logiciel

2.1 Contraintes

À la demande de notre tuteur, M. Lenté, la simulation a pris la forme d'une **simulation à événements discrets**, et a été développée en **Java**.

Tandis que la seconde contrainte ne concerne que la technologie, la première contrainte à grandement déterminée l'architecture du système.
En effet, une simulation à événements discrets doit respecter une forme stricte.

2.2 Structure de la simulation

Comme pour toute simulation, on cherche à suivre l'évolution d'un **système**, ici un ensemble de villages, de mots et de vocabulaires, articulé autour d'**événements** : émission d'un mot par un village, réception d'un mot par un village, mémorisation d'un mot par un village, élimination d'un mot du vocabulaire d'un village, événement initial, événement final.
Les **variables** du système qui nous intéressent sont les états des vocabulaires des différents villages.

Pour généraliser le problème, nous avons décidé de ne plus considérer de villages, mais des individus, d'appeler leur vocabulaire lexique, et de désigner par lemmes les mots qui le compose.

2.3 Système

Notre système est composé de n individus, indexés respectivement de 1 à n, et couramment désignés par la lettre de l'alphabet de leur indice.
Par exemple, le premier individu, d'indice 1, sera désigné par la lettre A.
Ces individus ont la capacité d'émettre, recevoir et mémoriser des lemmes, qui forment leur lexique respectif

Initialement, chaque individu connaît mi lemmes lui étant propres.
Ces lemmes sont formés de la lettre de l'individu et de l'indice du mot parmi le lexique d'origine.
Par exemple, si $M_1 = 2$*, l'individu A connaîtra initialement les mots A1 et A2.*

De plus, un individu ne peut jamais posséder plus de M_i lemmes dans son lexique (avec naturellement, $M_i \geq m_i$).

Enfin, chaque individu à seulement connaissance d'un sous-ensemble d'individus du système, l'ensemble de ses voisins, noté V_i.
À chacun de ses individus voisins est associé une valeur entière de délais T_i représentant la durée pour atteindre ce dernier, considérée notamment lors de l'émission d'un lemme.
Attention, la notion de voisin n'est pas bidirectionnelle.
Par exemple, l'individu A peut avoir comme voisin l'individu B avec un délais associé de valeur 2, alors que l'individu B peut n'avoir aucun voisin.

2.4 Évènements

Comme mentionné précédemment, l'évolution du système se fait exclusivement autour d'évènements.
Chaque fois qu'un événement est généré, il est stockés dans l'échéancier du système.
Dans l'échéancier, les événements sont rangés selon leur date effective par ordre croissant.

Voici la liste des différents évènements de la simulation :

- $E_{initial}$: événement initial, relatif au système, chargé de générer le premier événement d'émission [de lemme] depuis un individu du ;
- $E_{emission}$: événement d'émission [de lemme], relatif à un individu, chargé de générer des événements de réception [de lemme] depuis ses voisins et un événement d'émission [de lemme] depuis un voisin successeur ;
- $E_{reception}$: événement de réception [de lemme], relatif à un individu, chargé de générer un événement de mémorisation ;

— $E_{memorisation}$: événement de mémorisation [de lemme], relatif à un individu, chargé de générer un événement d'élimination [de lemme] dans le cas où le lexique de l'individu courant serait plein ;
— $E_{elimination}$: événement d'élimination [de lemme], relatif à un individu
— E_{final} : évènement final, relatif au système, généré selon une condition d'arrêt C_{arret}, concluant la simulation.

2.5 Déroulement de la simulation

La simulation commence par la génération d'un événement initial $E_{initial}$, et se finie lorsque la condition d'arrêt C_{arret} est satisfaite. À la fin de chaque événement, le prochain événement de l'échéancier est déclenché, permettant ainsi la succession des événements selon l'ordre respectif.

En effet, les événements du système ne sont pas nécessairement générés dans l'ordre chronologiques du système.

2.6 Conditions

Pour permettre une modélisation plus aboutie, l'émission, la réception et la mémorisation d'un lemme ne sont pas systématiques, et dépendent chacune d'une condition.

Ainsi, un lemme n'est émis avec succès que si la condition d'émission $C_{emission}$ de l'émetteur est satisfaite, puis ne peut être reçu par ses destinataires que si leur condition de réception $C_{reception}$ respective est vérifiée et enfin ne peut être mémorisé par ces derniers que si leur condition de mémorisation $C_{memorisation}$ est elle aussi vérifiée.

2.7 Stratégies

De plus, différentes stratégies sont mis en place pour étudier l'influence de certaines variables sur l'évolution du système.

Par exemple, la stratégie de sélection pour émission $S_{selection\ emission}$ adoptée par un individu détermine quel lemme émettre en fonction de son état actuel ou via une expérience aléatoire. De manière similaire, la stratégie de sélection pour élimination $S_{selectionelimination}$ d'un individu permet de déterminer quel lemme oublier lorsqu'un nouveau lemme est mémorisé mais que le lexique est plein. Enfin, la stratégie de succession $S_{succession}$ détermine pour un individu, une fois son lemme émis, quel voisin prendra sa succession.

2.8 Occurrence de lemme

Dans l'objectif d'analyser au mieux le déroulement de la simulation et d'en étudier l'évolution, on souhaite, à n'importe quel moment, être capable de reproduire l'état du système à une date antérieur. Or, par définition, un événement ne contient aucun donnée mais représente uniquement un déroulement d'actions. Pour pallier à ce problème, on va stocker les occurrences des lemmes pour chacun des événements (émission, réception, mémorisation, élimination) et ainsi garder une trace de leur conséquence sur le système.

2.9 Configuration du système

L'intérêt de la mise en place d'une simulation par informatique est de pouvoir choisir les différents paramètres à adopter, et par conséquent observer leur influence sur l'évolution du système.

En effet, il peut être intéressant de mettre en parallèle le nombre d'individus du système, la disposition du voisinage, les tailles de lexiques et les implémentations de conditions/stratégies par défaut pour des individus avec le résultat de la simulation pour pouvoir déterminer le rôle et l'importance des paramètres choisis.

De plus, permettre d'attribuer à certains individus du système des paramètres différents des autres est plus représentatif de la situation réelle, où chaque individu possède des spécificités qui lui sont propres, et permet donc une simulation plus fidèle à la réalité.

3 Modélisation du logiciel

3.1 Architecture générale

Notre application a été conçue entièrement en Java et suit le patron de conception **MVC** (Modèle Vue Contrôleur) en utilisant la plateforme **JavaFX**.

Chronologiquement, nous avons commencé par mettre en place le code métier pour la simulation en affichant les résultats du déroulement dans la console, puis, une fois la simulation fonctionnelle, nous avons développé une IHM pour mieux exploiter les résultats.

La Figure 1 décrit grossièrement comment se répartissent les rôles Modèle - Vue - Contrôleur dans notre architecture actuelle.
Pour connaître l'ensemble des classes, consulter la sous-section 4.

3.2 IHM

Comme dit précédemment, nous avons développé notre IHM en utilisant **JavaFX**.

Nous avons utilisé le logiciel **SceneBuilder** pour faciliter la création du fichier .fxml - le fichier à l'origine de l'interface graphique.
SceneBuilder étant un logiciel WYSIWYG (*What You See Is What You Get*), il a permi de grandement faciliter le design de l'IHM, qui aurait été très pénible autrement.

Nous avions initialement prévu un mockup pour notre IHM (figures 15, 16 et 17 en annexe), qui a été complété pour donner une version finale.

Dans sa version finale (figures 2, 3 et 4), l'utilisateur peut :

- Consulter le voisinage du système ;
- Pour un individu en particulier ou l'ensemble des individus, consulter :
  - La composition finale de son/leur lexique ;
  - L'ensemble des évènements le/les concernant ;
  - Ses/leurs paramètres de configuration ;
  - L'évolution de son/leur lexique dans le temps ;
  - L'évolution de ses/leurs occurrences de lemmes.

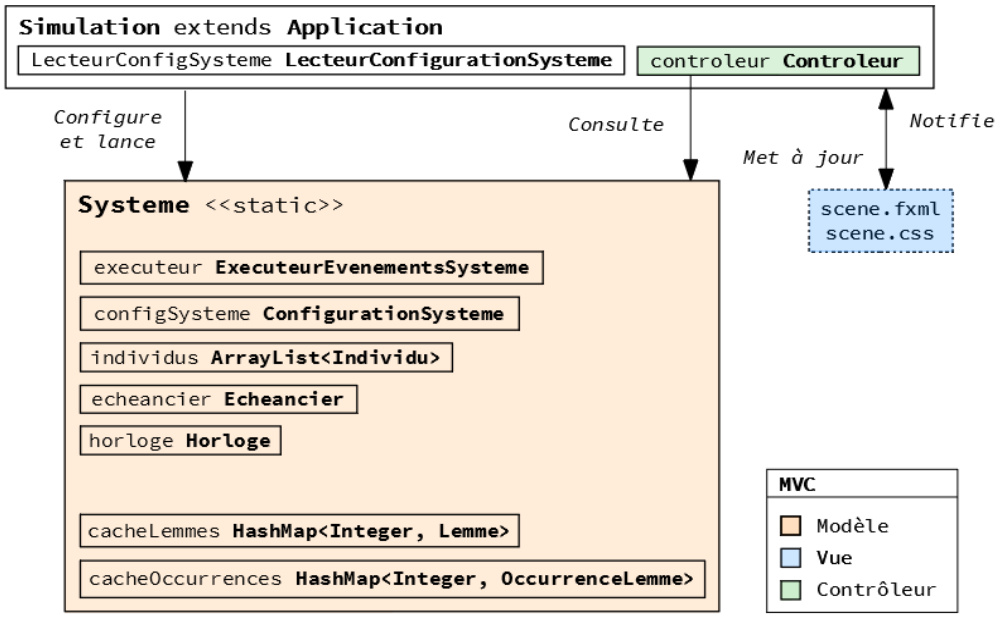

Figure 1 – *Architecture simplifiée du logiciel*

Figure 2 – *IHM de l'application (1)*

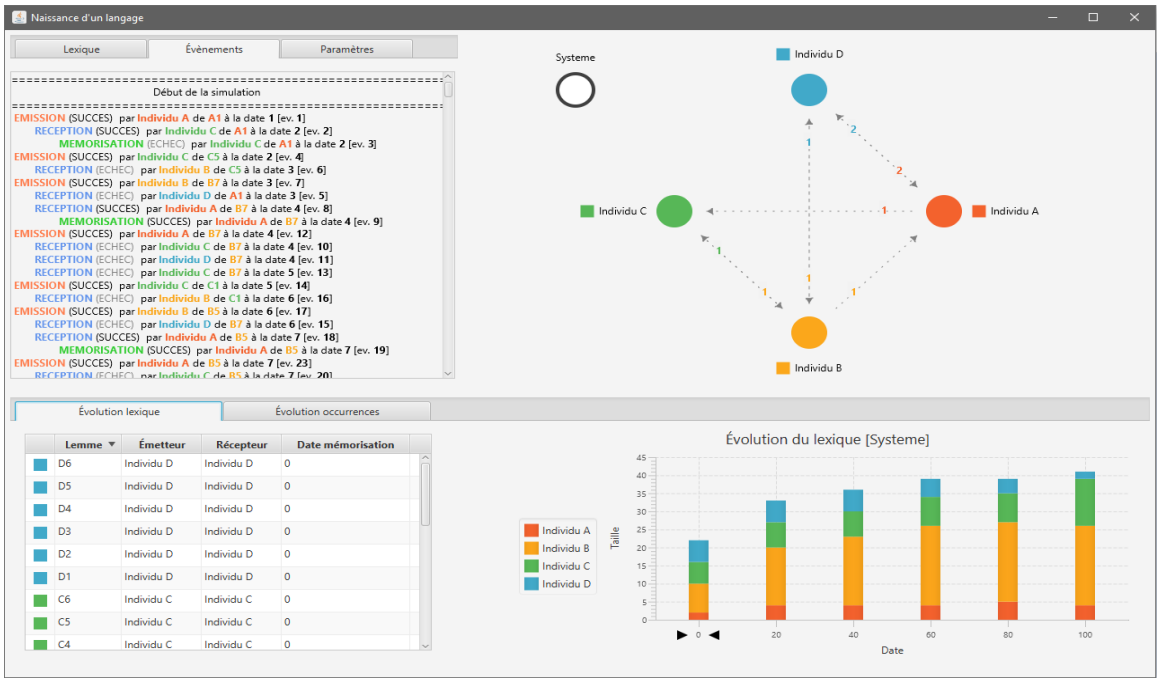

Figure 3 – *IHM de l'application (2)*

Figure 4 – *IHM de l'application (3)*

4 Implémentation du logiciel

4.1 Fichiers sources

Notre projet prend la forme de 50 fichiers sources, organisé en 20 packages.
Son diagramme de classes épuré est décrit à travers la figure 5.
Les packages les plus importants, ainsi que leurs rôles, sont :

- **simulation.ihm**, qui contient les classes Controleur, Portee et ses implémentations. Ces sont les principales de l'IHM, qui permettent de faire le lien entre le modèle de la simulation (le système) et l'interface graphique.
- **simulation.systeme**, qui contient notamment les classes Systeme et Individu, autrement dit le coeur de la simulation. Ce package contient des sous-packages spécialisés comme :
  - **simulation.systeme.lexique**, qui contient les classes relatives aux lemmes ;
  - **simulation.systeme.temps**, qui contient les classes manipulant des donnés temporelles (Date et Delais) ;
  - **simulation.systeme.evenement**, qui contient la classe Echeancier, qui contient les évènements en attente, et la classe Evenement avec ses implémentations (EvenementInitial, EvenementEmission, etc.), utilisées à travers les classes ExecuteurEvenementSysteme et ExecuteurEvenementIndividu ;
  - **simulation.systeme.condition**, qui contient la classe abstraite Condition et ses implémentations (ConditionToujoursVerifiee, etc.) ;
  - **simulation.systeme.strategie**, qui contient les classes abstraites StrategieSelection et StrategieSuccession, et leurs implémentations (StrategieSelectionLemmeAleatoire, StrategieSuccessionPremierVoisin, etc.).

4.2 Fichiers de configuration

La configuration de la simulation repose sur 2 fichiers : **config.xml** et **matrice.txt**.

config.xml

Le premier fichier de configuration, config.xml (disponible en annexe à la page 12), indique la plupart des paramètres pour la simulation, notamment :

- Le nombre d'individus du système ;
- La taille du lexique initiale et finale par défaut pour les individus ;
- Le chemin vers le fichier graphe.txt ;
- Les implémentations pour les conditions et stratégies par défaut pour les individus ;
- Le critère d'arrêt de la simulation. Mais aussi :
- La taille du lexique initiale et finale pour un individu en particulier ;
- Les implémentations pour les conditions et stratégies pour un individu en particulier.

Le schéma de définition (*XML Schema Definition*) suivi par le fichier de configuration est également disponible en annexe, à la page 13.
Ce schéma n'est a priori pas conforme avec la norme W3C XSD 1.1, mais permet tout de même à une personne connaissant la norme de connaître les valeur acceptables.

matrice.txt

Le fichier matrice.txt (disponible en annexe à la page 12) , quant à lui, contient l'organisation du voisinage des individus sous la forme d'une matrice.

De manière semblable à la matrice d'adjacence d'un graphe, la matrice de voisinage du système indique pour chaque individu le délais associé à ses voisins.

Soit M la dite matrice, de taille nxn.
$M[i][j] > 0$ signifie que l'individu d'ID i a comme voisin l'individu d'ID j avec un délais de valeur $M[i][j]$. Sinon, l'individu d'ID i est considéré comme n'ayant pas l'individu j comme voisin.

Afin de garantir que la simulation ne se trouvera pas dans une situation anormale prévisible, comme par exemple avec un individu totalement isolé du reste du système, des contraintes sont imposées sur le voisinage :

— Un individu doit avoir au moins un voisin, i.e une ligne ne peut pas être nulle ;
— Un individu ne peut être son propre voisin, i.e. $M[i][i] = 0 \forall i$;
— Un délais ne peut pas être négatif, i.e $M[i][j] >= 0 \forall i$.

4.3 Fichiers de tests

Comme notre application de simulation exige le travail de nombreuses classes simples mêlée entre elles d'une manière assez complexe, c'est pourquoi mettre en place des tests unitaires pour chaque entité n'avait pas vraiment d'intérêt.
De plus, l'architecture générale a beaucoup évoluée et implémenter des tests trop tôt n'aurait pas été très judicieux.

Ainsi, dans un premier temps, nous avons détecté les erreurs de programmation en vérifiant la cohérence des résultats en fonctions des données.

Une fois que nous avons obtenu une version stable, nous avons mis en place deux fichiers de tests avec **JUnit 4** : l'un pour vérifier le bon fonctionnement du parser de configuration système, **LecteurConfigurationTest.java**, et l'autre pour le système dans son ensemble, **SystemeTest.java**.

Le premier utilise de la paramétrisation (package org.junit.runners.Parameterized) et ainsi facilite grandement l'ajout de scénarios de tests.

Désormais, dans le cas où nous ferions une modification, il nous est possible de vérifier le bon fonctionnement du programme en lançant ces tests.

5 Résultats et analyse

Lorsqu'on étudie l'évolution d'un langage, on peut diffcilement ne pas avoir recourt à des notions de **Théorie des graphes**.

En effet, comme la langue Européenne est totalement différente de celle d'Asie, l'organisation spatiale d'un système est déterminante pour l'évolution d'un langage. C'est donc assez naturellement que l'on modélise ce genre de phénomène par un graphe.

Un graphe est un modèle mathématique abstrait utilisé dans la représentation de réseaux.

Pour notre simulation, le graphe représente le système, où chaque sommet représente un individu et chaque arc représente une relation de voisinage.

Un graphe possède comme caractéristique principale sa **topologie**, qui représente son organisation spatiale générale. Il existe 3 classes de topologies. Les topologies dites **structurés** (homogènes, hiérarchiques, cycliques et centralisés/polaires), qui suivent des modèles remarquables, les topologies dites **quelconques**, qui au contraire, n'ont aucune propriété particulière, et pour finir les topologies **multipolaires**.

5.1 Analyse topologique

Dans la partie suivante, nous allons étudier l'influence des différentes topologies sur le déroulement de la simulation.

Pour la suite, la configuration système considérée est :

— tailleIntialeLexiqueParDefaut : 6 ;

— tailleMaximaleLexiqueParDefaut : 12 ;
— implConditionReceptionParDefaut : CONDITION_PROBABILITE_UNIFORME ;
— implConditionEmissionParDefaut : CONDITION_TOUJOURS_SATISFAITE ;
— implConditionReceptionParDefaut : CONDITION_PROBABILITE_UNIFORME ;
— implConditionMemorisationParDefaut : CONDITION_PROBABILITE_UNIFORME ;
— implStrategieSelectionEmissionParDefaut : SELECTION_LEMME_ALEATOIRE ;
— implStrategieSelectionEliminationParDefaut : SELECTION_LEMME_ALEATOIRE ;
— implStrategieSuccessionParDefau : SUCCESSION_VOISIN_ALEATOIRE ;
— implCritereArret : TOUS_LEXIQUES_PLEINS ;
— Individu spécifique : Aucun.

Graphes structurés [4]

Parmi les graphes structurés, on note 4 structures remarquables :

— **Graphes homogènes** : les sommets et les arêtes reproduisent un schéma régulier. Le schéma le plus commun est une architecture de type matriciel aussi appelé "en filet de poisson" (mesh) ;
— **Graphes hiérarchiques** : structure typique des graphes où les sommets s'arrangent en couches hiérarchisées et pyramidales ;
— **Graphes cycliques** : on peut identifier des cycles dans le graphe. L'exemple le plus parlant est le graphe circulaire ;
— **Graphes centralisés/polaires** : C'est une architecture où tous les sommets sont rattachés à un seul sommet, le pôle.

Graphe homogène

Pour un graphe de type homogène (figure 6), on note que le pourcentage du lexique de système reste toujours équilibré de l'état initale à l'état final lors de la simulation.

Graphe hiérarchique

En accord avec la topologie hiérarchique (figure 7), l'individu B et l'individu C ont un rôle important du point de vue du système puisqu'ils sont **en haut de l'arborescence**. Les individus suivants en sont grandement dépendants.

Graphe cyclique

Comme en témoigne le diagramme de composition finale du lexique ((figure 8), un graphe cyclique a tendance à **équilibrer le réseau** entre les différents individus.

Graphe centralisé

En théorie des graphes et en analyse de réseau, les indicateurs de **centralité** identifient les sommets les plus importants d'un graphe. Cette notion s'applique notamment dans l'identification des personnes les plus influentes d'un réseau social, des nœuds d'infrastructure clés sur Internet ou des réseaux urbains et des super-propagateurs de maladies. Dans cette expérimentation (figure 9), on considères toutes les occurrences de lemmes d'issue "Succès". D'après les résultats, il est clair que l'individu A, central au système, a un très grand impact sur le reste des individus.

Graphe quelconque

Pour la topologie quelconque, comme en témoigne la figure 10, aucune propriété topologique ne semble émerger.

Graphe multipolaire

L'architecture multipolaire est très intéressante. C'est une architecture mixte entre les graphes centralisés et décentralisés. Les réseaux multipolaires sont très étudiés en raison de leur similitude avec de nombreux cas concrets (Internet, réseaux neuraux, etc.). Les graphes multipolaires sont caractérisés par deux types d'arêtes, celles qui forment les liens émanant du pôle, les liens forts ; et les liens réunissant deux pôles entre eux : les liens faibles. Les pôles peuvent par ailleurs prendre une architecture structurée (souvent centralisée) ou quelconque[4].

D'après la topologie suivie (figure 11), l'individu A représente un **pôle** avec B, C, D et F, et I représente un plus petit pôle avec H, E et G.
Le **lien faible** entre ces deux pôles est composé par D et H. Dans ce graphe-ci, il n'y a pas de sommet solitaire.
Comme nous l'avons déjà souligné avec le graphe centralisé, le centroïde de chaque pôle a un grand impact dans son pôle, et le lien entre deux pôles a un rôle d'échangeur.

Enfin, on pourrait voir le graphe de la façon suivante : les individus D et H sont des **traducteurs**, et les pôles A et I sont comme deux **pays** parlant une langue différente.

D'après les résultats ci-dessus, on a observé une réelle influence de la topologie du système sur son évolution. On se propose désormais d'observer l'influence des critères quantitatifs (nombre d'individus, tailles initiale/maximale de lexique).

5.2 Analyse des tailles

On se propose désormais d'analyser l'influence des **critères numériques** sur l'évolution du système, plus particulièrement sur la vitesse de remplissage des lexiques des individus.

Par critères numéirques, nous entendons les suivants :

- **Le nombre de sommets** : Il s'agit du nombres d'individus dans le système ;
- **Les valeurs de délais/distances entre individus** : La distance entre individus est un facteur évident, puisqu'elle implique un délais qui retarde certaines communications ;
- **Le degré du graphe** : C'est le nombre d'arcs partants et arrivants de chaque sommets, ce qui correspond dans notre simulation aux liaison de voisinage des individus. ;
- **Le niveau de connaissance (La taille initiale/finale des lexiques)** : Comme leur nom l'indique, il s'agit du nombre de lemmes connus au début de la simulation par chaque individus et le nombre de lemmes maximum que ces derniers peuvent mémoriser.

Pour la suite de cette partie, nous considérerons un système de configuration par défaut suivante, bien que nous fassions varier certains d'entre eux :

- Nombre d'individus : 9 ;
- Topologie du graphe du système : cyclique ;
- TailleIntialeLexiqueParDefaut : 3 ;
- TailleMaximaleLexiqueParDefaut : 20 ;
- ConditionReceptionParDefaut : CONDITION_PROBABILITE_UNIFORME ;
- ConditionEmissionParDefaut : CONDITION_TOUJOURS_SATISFAITE ;
- ConditionReceptionParDefaut : CONDITION_PROBABILITE_UNIFORME ;

— ConditionMemorisationParDefaut : CONDITION_PROBABILITE_UNIFORME ;
— StrategieSelectionEmissionParDefaut : SELECTION_LEMME_ALEATOIRE ;
— StrategieSelectionEliminationParDefaut : SELECTION_LEMME_ALEATOIRE ;
— StrategieSuccessionParDefau : SUCCESSION_VOISIN_ALEATOIRE ;
— CritereArret : TOUS_LEXIQUES_PLEINS ;
— Individu spécifique : Aucun.

Analyse du nombre de sommets/délais

On se propose ici de faire varier le nombre d'individus ainsi que la valeur de délais entre les individus de 3 à 9 et d'observer les résultats.

La figure 12 montre que le délais (courbe rouge), contrairement au nombre d'individus (courbe bleue), a une grande influence sur la vitesse à laquelle les individus remplissent leur lexique. En effet, les délais étant présents à la fois à la réception et à la succession, l'évolution de l'ensemble du système est considérablement retardée par les délais de voisinage.

Analyse du degré de graphe

En mathématiques, et plus particulièrement en théorie des graphes, le degré (ou valence) d'un sommet d'un graphe est le nombre de liens (arêtes ou arcs) reliant ce sommet. Et le degré ici représente l'intensité de graphe. Plus le degré est élevé, plus la relation entre les sommets est étroite.

Pour profonder depuis cet aspect, nous faisons désormais varier le degré du graphe de 3 à 7, c'est-à-dire considérer pour chaque individu de 3 à 7 voisins.

La figure 13 illustre bien le fait que plus la relation entre les individus est étroite, plus les lemmes se répendent rapidement et par conséquent plus les lexiques des individus du système se remplissent rapidement.

Analyse du niveau de connaissance

Observons désormais l'influence du niveau de connaissance (i.e les taille initiale et finale de lexiques) sur le remplissage des lexiques du système. Pour le graphe suivant (Figure 14), on fait prendre les valeurs 3, 6, 12 et 15 à la taille initiale de lexique avec une taille maximale fixe de 20, et les valeurs 5, 8, 15 et 20 à la taille maximale de lexique avec une taille initiale fixe de 3.

Il est assez clair par la figure 14 que les individus d'un système remplissent rapidement leur lexique s'ils connaissent initialement un grand nombre de lemmes (courbe bleue, à droite) ou si leur lexique possède une taille maximale petite (courbe rouge, à gauche).

6 Conclusion

Pour aller plus loin

Nous avons envisagé certaines améliorations pour notre simulation, afin de la rendre beaucoup plus cohérente avec le contexte.
Ces améliorations concernent la complexité des lemmes et les stratégies de sélection.

En effet, les lemmes de chaque individus sont actuellement formés selon un algorithme beaucoup trop basique : la lettre de l'individu à laquelle on concatène l'indice du lemme dans le lexique. Or, une langue est un ensemble riche et hétérogène de lemmes, qui peuvent être de tailles différentes et être plus ou moins complexes.

Un tel critère n'est pas négligeable, puisqu'il est assez clair qu'un mot compliqué aura tendance à être rarement employé, alors qu'un mot simple sera beaucoup plus commun.

De même, un individu recevant un lemme semblable à un qu'il connaît déjà aura sans doute plus de facilité pour le retenir.

Pour les raisons précédentes, mettre en place une stratégie de génération (StrategieGeneration par exemple) suivant un schéma plus proche de celui des langues actuelles, une implémentation de stratégie de sélection de lemme pour les lemmes simples (ImplStrategieSelectionLemmeSimple) ou une implémentation de condition pour mémorisation les lemmes semblables (ImplConditionLemmeSemblable) peut être très intéressant.

Beaucoup d'autres critères pourraient être considérés dans la simulation, comme par exemple prendre en compte les syllabes dans la complexité des lemmes, faciliter la mémorisation de lemmes déjà reçus mais qui n'ont pas été mémorisés ou dont on connaît déjà plusieurs autres lemmes du même initiateur, etc.

Enfin, la parallélisation des évènements d'émission permettrait d'avoir une évolution du système plus dynamique et rapide, et réduirait l'impact des délais sur l'avancement global de la simulation, comme observé dans la sous-section d'analyse 5.2.

7 Annexe

config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Schema de definition disponible sous schema.xsd -->
<config>
    <!-- Configuration du systeme -->
  <systeme nombreIndividus="4" tailleInitialeLexiqueParDefaut="6"
  tailleMaximaleLexiqueParDefaut="12"
  urlFichierMatriceVoisinage="config/matrice.txt">
    <implConditionEmissionParDefaut>CONDITION_TOUJOURS_SATISFAITE
    </implConditionEmissionParDefaut>
    <implConditionReceptionParDefaut>CONDITION_PROBABILITE_UNIFORME
    </implConditionReceptionParDefaut>
    <implConditionMemorisationParDefaut>CONDITION_PROBABILITE_UNIFORME
    </implConditionMemorisationParDefaut>
    <implStrategieSelectionEmissionParDefaut>SELECTION_LEMME_ALEATOIRE
    </implStrategieSelectionEmissionParDefaut>
    <implStrategieSelectionEliminationParDefaut>SELECTION_LEMME_ALEATOIRE
    </implStrategieSelectionEliminationParDefaut>
    <implStrategieSuccessionParDefaut>SUCCESSION_PREMIER_VOISIN
    </implStrategieSuccessionParDefaut>
    <typeCritereArret>TOUS_LEXIQUES_PLEINS</typeCritereArret>
  </systeme>

  <!-- Configuration d individus specifiques -->
  <individu id="1" tailleInitialeLexique="2" tailleMaximaleLexique="5">
    <implConditionEmission>CONDITION_TOUJOURS_SATISFAITE</implConditionEmission>
    <implConditionReception>CONDITION_TOUJOURS_SATISFAITE</implConditionReception>
    <implConditionMemorisation>CONDITION_TOUJOURS_SATISFAITE</implConditionMemorisation>
    <implStrategieSelectionEmission>SELECTION_PREMIER_LEMME</implStrategieSelectionEmission>
  </individu>
  <individu id="2" tailleInitialeLexique="8"></individu>
</config>
```

matrice.txt

```
# Matrice de voisinage
#
```

```
# Si M[i][j] > 0, l'individu d'ID i a comme voisin l'individu d'ID j avec un delais de valeur M[i][j]
# Sinon, l'individu d'ID i n'a pas l'individu j comme voisin
#
# Notes:
# - La matrice doit carree de taille N, avec N egal a l'attribut 'nombreIndividus' du fichier de config systeme;
# - Un individu doit avoir au moins un voisin, i.e une ligne ne peut pas etre nulle;
# - Un individu ne peut etre son propre voisin, i.e. M[i][i] = 0 pour tout i;
# - Un delais ne peut pas etre negatif, i.e M[i][j] >= 0 pour tout i.
#
0 0 1 2
1 0 1 1
0 1 0 0
2 1 0 0
```

schema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Structure de l element <config> -->
  <xs:element name="config">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="systeme" type="Systeme" use="required"/>
        <xs:element name="individu" type="Individu" use="optional"/>
      </xs:sequence>
    </xs:complexType>

    <!-- Contrainte d unicite de l attribut id des elements individu -->
    <xs:unique name="IndividuID">
      <xs:selector xpath="individu"/>
      <xs:field xpath="@id"/>
    </xs:unique>
  </xs:element>

  <!-- Structure de l element <systeme> -->
  <xs:complexType name="Systeme">
    <xs:attribute type="One-to-ten" name="nombreIndividus" use="required"/>
    <xs:attribute type="One-to-twenty" name="tailleInitialeLexiqueParDefaut" use="required"/>
    <xs:attribute type="One-to-twenty" name="tailleMaximaleLexiqueParDefaut" use="required"/>
    <xs:attribute type="xs:string" name="urlFichierGraphe" use="required"/>
    <xs:element type="ImplementationCondition" name="implConditionEmissionParDefaut" use="required"/>
    <xs:element type="ImplementationCondition" name="implConditionReceptionParDefaut" use="required"/>
    <xs:element type="ImplementationCondition" name="implConditionMemorisationParDefaut" use="required"/>
    <xs:element type="ImplementationStrategieSelection" name="implStrategieSelectionEmissionParDefaut" use="required"/>
    <xs:element type="ImplementationStrategieSelection" name="implStrategieSelectionEliminationParDefaut" use="required"/>
    <xs:element type="ImplentationStrategieSuccession" name="implStrategieSuccessionParDefaut" use="required"/>
    <xs:element type="TypeCritereArret" name="typeCritereArret" use="required"/>

    <!-- Contrainte sur la tailles initiale et maximale des lexiques -->
    <xs:assert test="@tailleInitialeLexiqueParDefaut &lt= @tailleMaximaleLexiqueParDefaut"/>
  </xs:complexType>

```

```
38  <!-- Structure de l element <individu> -->
39  <xs:complexType name="Individu">
40    <xs:attribute type="One-to-ten" name="id" use="required"/>
41    <xs:attribute type="One-to-twenty" name="tailleInitialeLexique" use="optional"/>
42    <xs:attribute type="One-to-twenty" name="tailleMaximaleLexique" use="optional"/>
43
44    <xs:element type="ImplementationCondition" name="implConditionEmission" use="optional"/>
45    <xs:element type="ImplementationCondition" name="implConditionReception" use="optional"/>
46    <xs:element type="ImplementationCondition" name="implConditionMemorisation" ↩
            use="optional"/>
47    <xs:element type="ImplementationStrategieSelection" ↩
            name="implStrategieSelectionEmission" use="optional"/>
48    <xs:element type="ImplementationStrategieSelection" ↩
            name="implStrategieSelectionElimination" use="optional"/>
49    <xs:element type="ImplentationStrategieSuccession" name="implStrategieSuccession" ↩
            use="optional"/>
50
51    <!-- Contrainte sur la valeur de l attribut id -->
52    <xs:assert test="@id &lt;= systeme@nombreIndividus"/>
53
54    <!-- Contrainte sur la valeur des atttributs tailleMaximaleLexique et ↩
            tailleMaximaleLexique -->
55    <xs:assert test=
56      "(not (@tailleInitialeLexique) and not(@tailleMaximaleLexique)) or
57      (@tailleInitialeLexique and @tailleMaximaleLexique and @tailleInitialeLexique &lt;= ↩
              @tailleMaximaleLexique) or
58      (@tailleInitialeLexique and (@tailleInitialeLexique &lt;= ↩
              systeme@tailleMaximaleLexiqueParDefaut)) or
59      (@tailleMaximaleLexique and (@tailleMaximaleLexique &gt;= ↩
              systeme@tailleInitialeLexiqueParDefaut))"
60    />
61  </xs:complexType>
62
63  <!-- Types utilises avec restrictions associees -->
64  <xs:simpleType name="One-to-ten">
65    <xs:restriction base="xs:integer">
66      <xs:minInclusive value="1"/>
67      <xs:maxInclusive value="10"/>
68    </xs:restriction>
69  </xs:simpleType>
70
71  <xs:simpleType name="One-to-twenty">
72    <xs:restriction base="xs:integer">
73      <xs:minInclusive value="1"/>
74      <xs:maxInclusive value="20"/>
75    </xs:restriction>
76  </xs:simpleType>
77
78  <xs:simpleType name="ImplementationCondition">
79    <xs:restriction base="xs:string">
80      <xs:enumeration value="CONDITION_PROBABILITE_UNIFORME"/>
81      <xs:enumeration value="CONDITION_TOUJOURS_SATISFAITE"/>
82      <xs:enumeration value="CONDITION_JAMAIS_SATISFAITE"/>
83    </xs:restriction>
84  </xs:simpleType>
85
86  <xs:simpleType name="ImplementationStrategieSelection">
87    <xs:restriction base="xs:string">
88      <xs:enumeration value="SELECTION_PREMIER_LEMME"/>
89      <xs:enumeration value="SELECTION_LEMME_ALEATOIRE"/>
90      <xs:enumeration value="SELECTION_LEMME_MOINS_EMIS"/>
```

```
91      </xs:restriction>
92    </xs:simpleType>
93
94    <xs:simpleType name="ImplentationStrategieSuccession">
95      <xs:restriction base="xs:string">
96        <xs:enumeration value="SUCCESSION_VOISIN_ALEATOIRE"/>
97        <xs:enumeration value="SUCCESSION_PREMIER_VOISIN"/>
98      </xs:restriction>
99    </xs:simpleType>
100
101   <xs:complexType name="CritereArret">
102     <xs:simpleContent>
103       <xs:extension base="TypeCritereArret">
104         <xs:attribute name="objectif">
105           <xs:simpleType>
106             <xs:restriction base="xs:integer">
107               <xs:minInclusive value="1"/>
108               <xs:maxInclusive value="2000"/>
109             </xs:restriction>
110           </xs:simpleType>
111         </xs:attribute>
112       </xs:extension>
113     </xs:simpleContent>
114   </xs:complexType>
115
116   <xs:simpleType name="TypeCritereArret">
117     <xs:restriction base="xs:string">
118       <xs:enumeration value="EVENEMENTS_DECLENCHES"/>
119       <xs:enumeration value="DATE_ATTEINTE"/>
120       <xs:enumeration value="PREMIER_LEXIQUE_PLEIN"/>
121       <xs:enumeration value="TOUS_LEXIQUES_PLEINS"/>
122     </xs:restriction>
123   </xs:simpleType>
124     <!-- dans les cas PREMIER_LEXIQUE_PLEIN et TOUS_LEXIQUES_PLEINS, l attribut objectif ↩
            n est pas obligatoire -->
125 </xs:schema>
```

[3] [5] [6] [2] [1] [7]

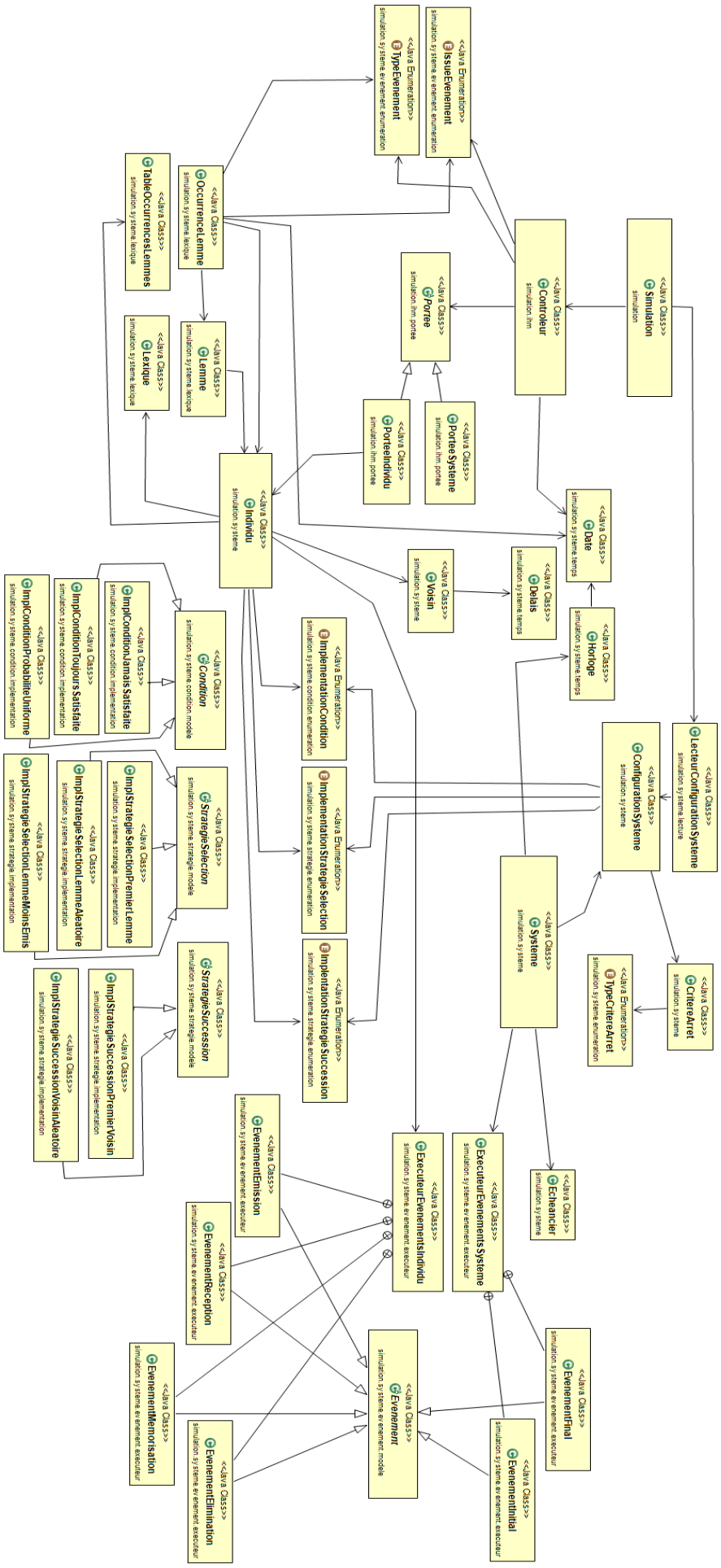

Figure 5 – *Diagramme de classe de l'application*

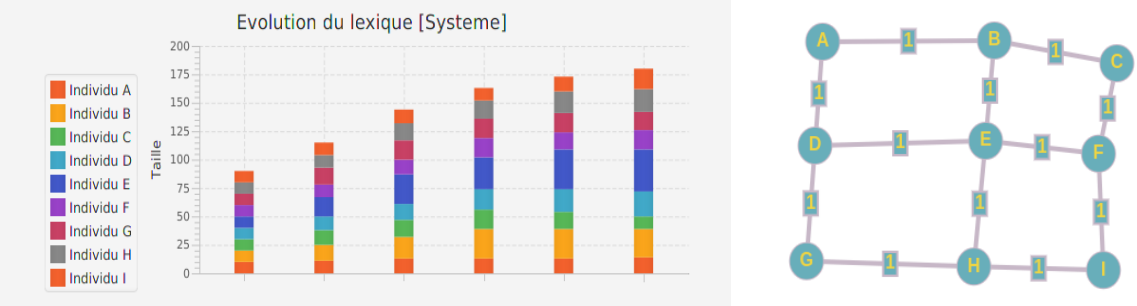

Figure 6 – *Évolution du lexique (IHM) et topologie d'un graphe homogène*

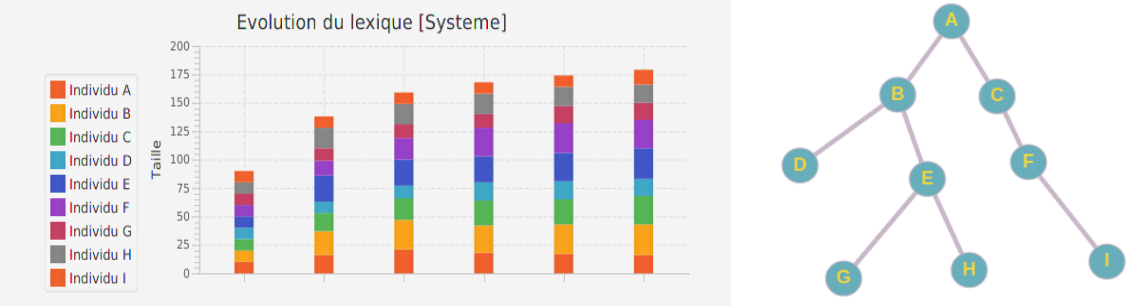

Figure 7 – *Évolution du lexique (IHM) et topologie d'un graphe hiérarchique*

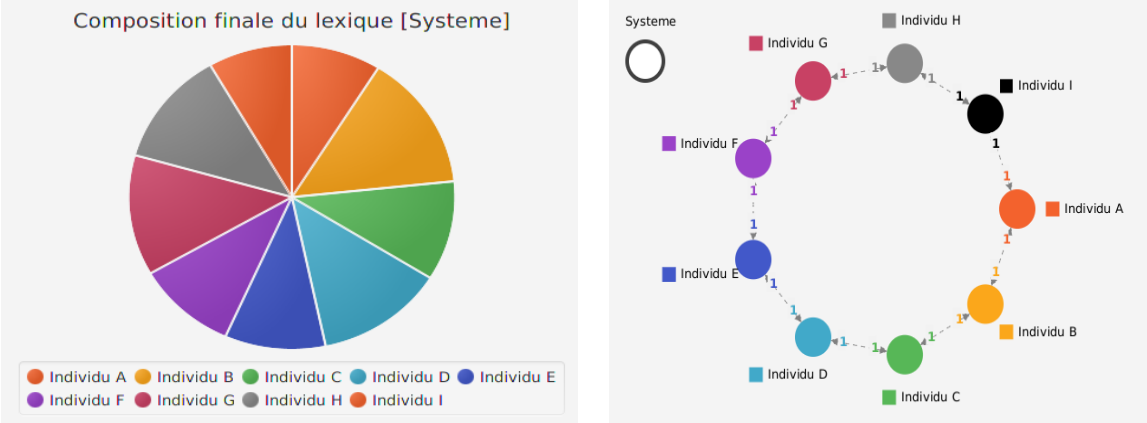

Figure 8 – *Composition finale du lexique (IHM) et topologie d'un graphe cyclique*

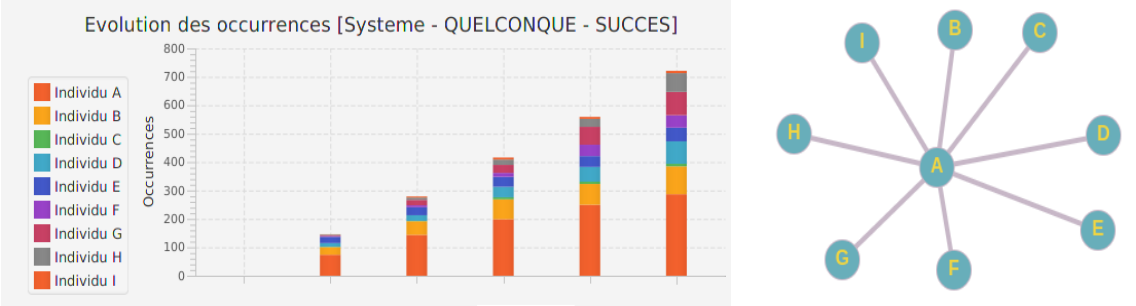

Figure 9 – *Évolution du lexique (IHM) et topologie d'un graphe centralisé*

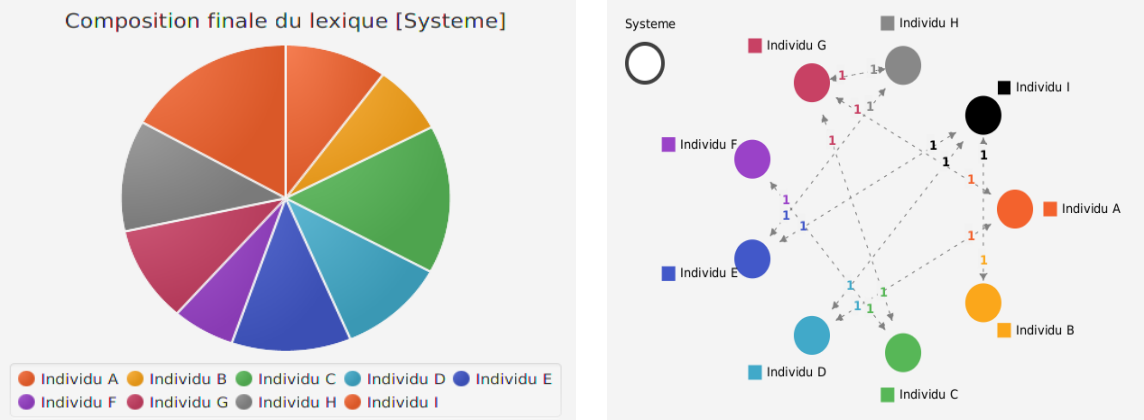

Figure 10 – *Composition finale du lexique et topologie d'un graphe quelconque*

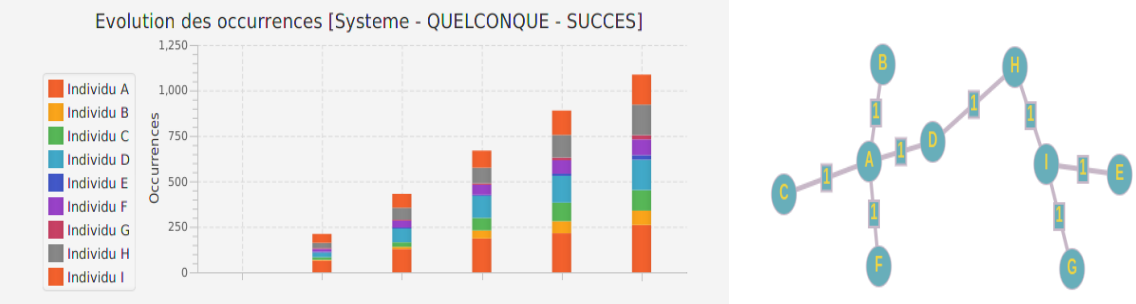

Figure 11 – *Évolution des occurrences (IHM) et topologie d'un graphe multipolaire*

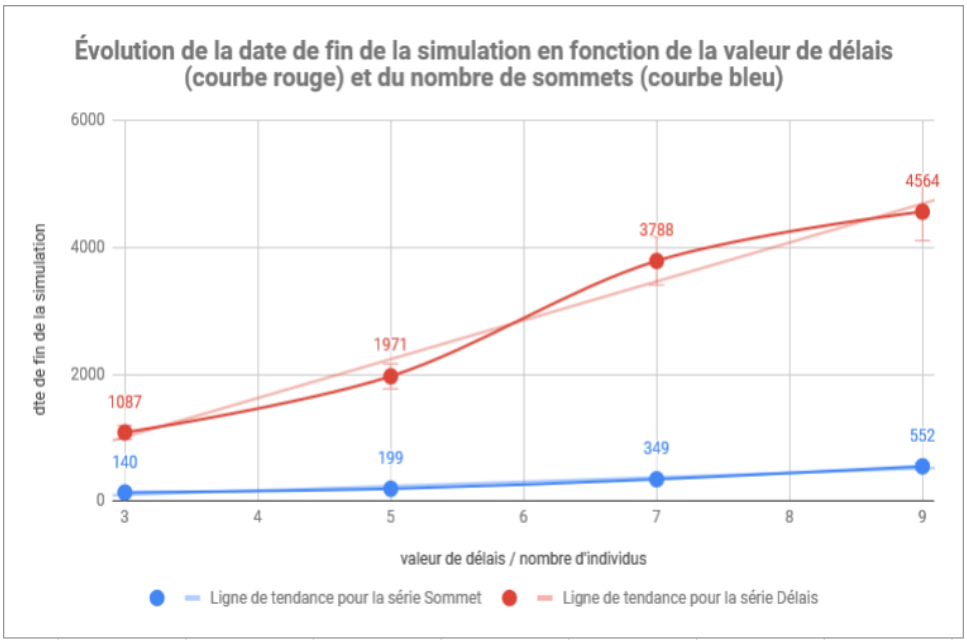

Figure 12 – *Impact du nombre d'individus et du délais sur la date finale de la simulation*

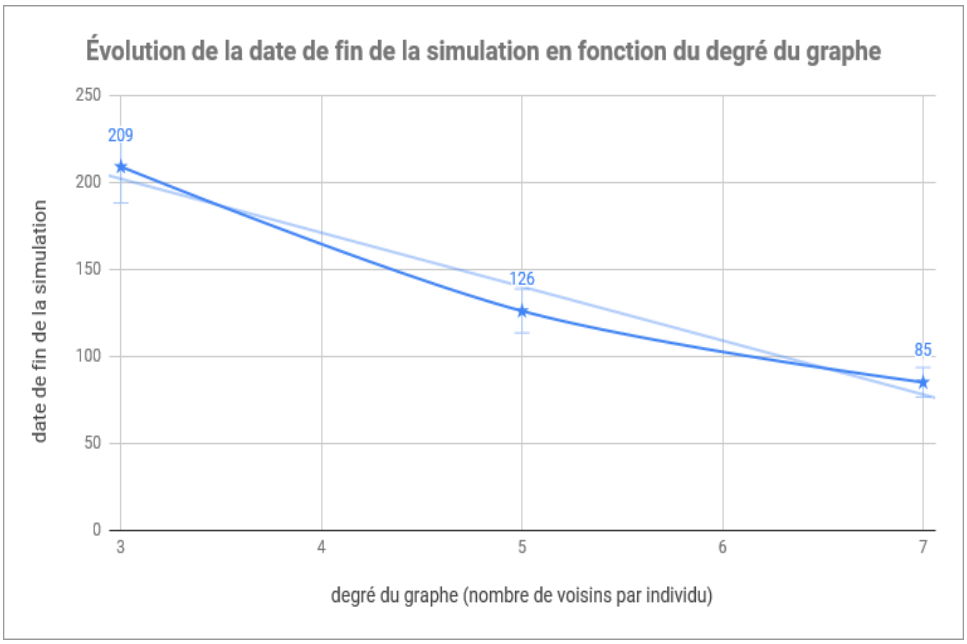

Figure 13 – *Impact du degré du graphe du système sur la date de fin de la simulation*

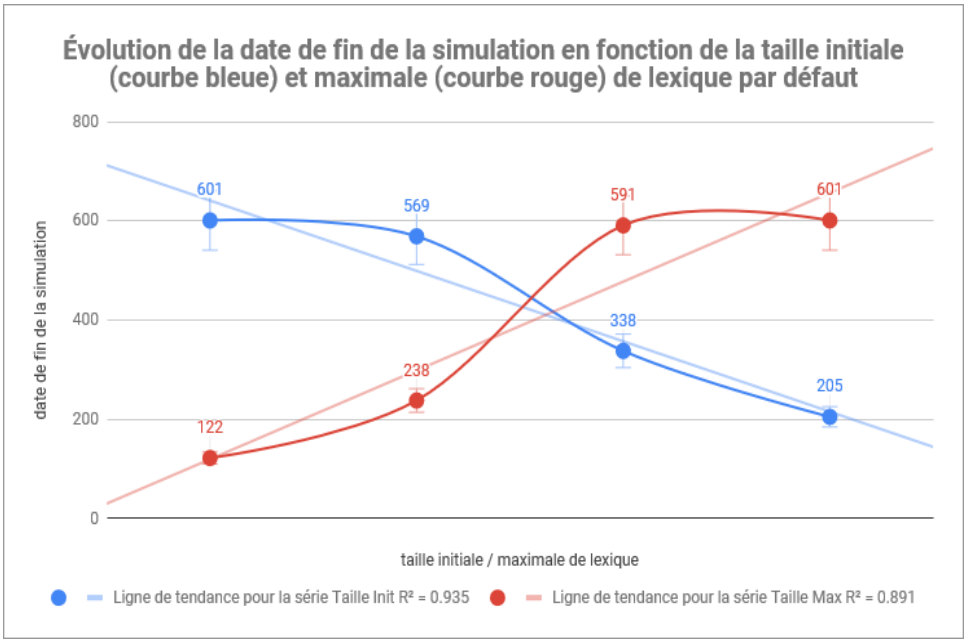

Figure 14 – *Impact des tailles de lexique sur la date de fin de la simulation*

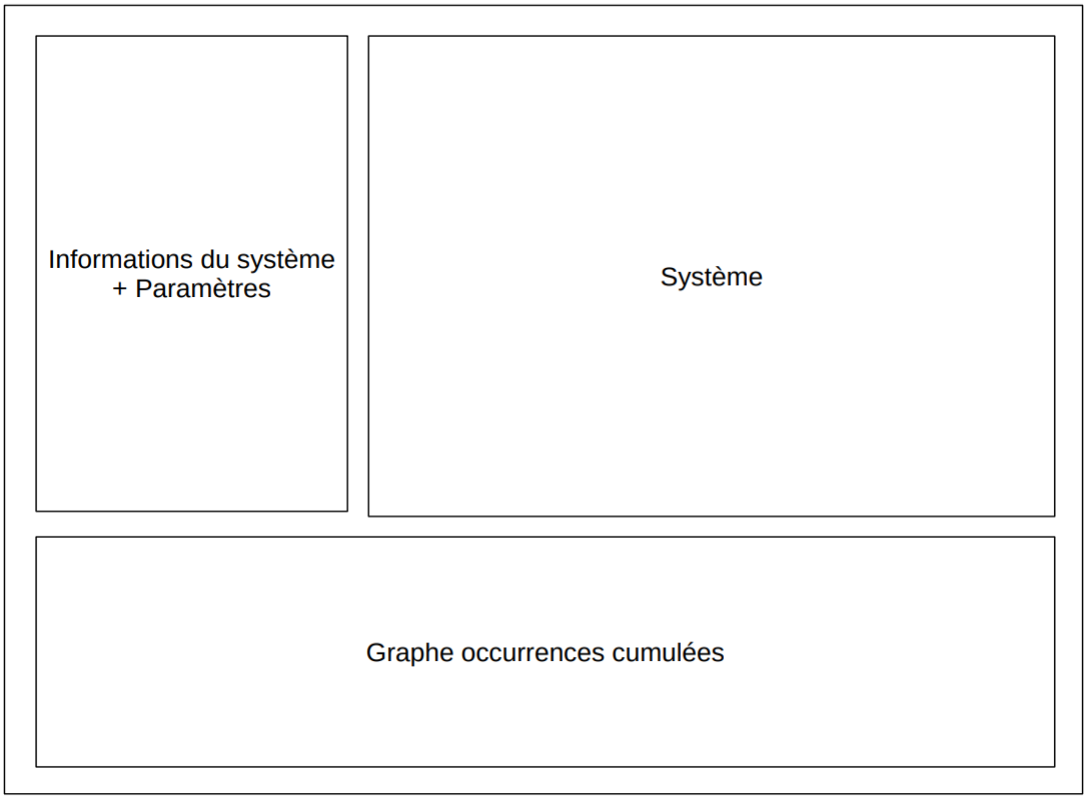

Figure 15 – *Mockup de l'IHM (1)*

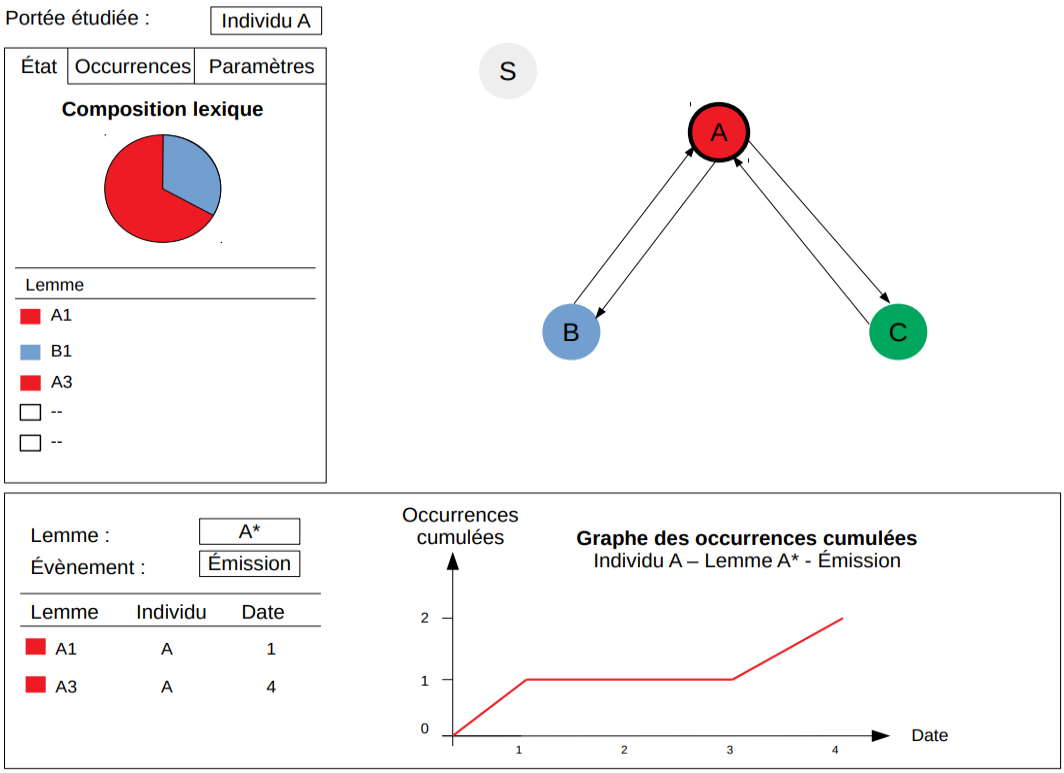

Figure 16 – *Mockup de l'IHM (2)*

Portée étudiée : Système || Indivdu [A || B || C]

Évènement : Émission || Réception || Mémorisation || Élimination

Lemme : [A || B || C] [1 || … || X] avec X signifiant “n’importe quel chiffre”

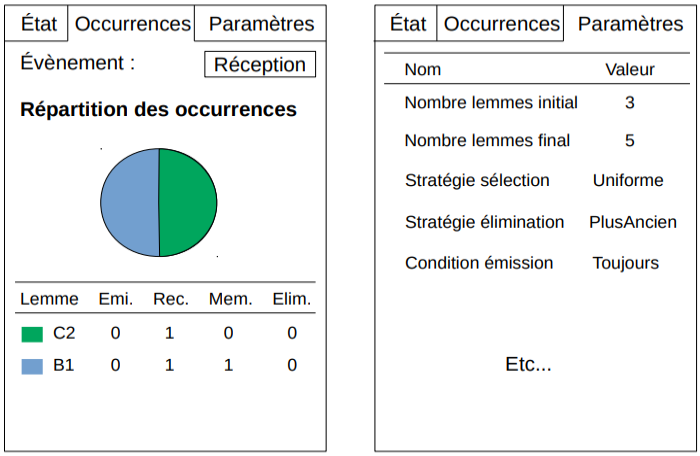

Figure 17 – *Mockup de l’IHM (3)*

Bibliographie

[1] Jean-Pierre Changeux. *Théories du langage et théories de l'apprentissage*. Seuil. 1979.

[2] Olivier Cogis et Claudine Schwartz. *Théorie des graphes*. ISBN 978-2-84225-189-5. 2018.

[3] Contributeur(s). *Wikipedia | Centralité*. https://fr.wikipedia.org/wiki/Centralité. Consulté le 8 janvier 2019. Nov. 2018.

[4] Contributeur(s). *Wikipedia | Théorie des Graphes*. https://fr.wikipedia.org/wiki/Théorie_des_graphes. Consulté le 8 janvier 2019. Nov. 2018.

[5] Contributeur(s). *Wikipedia | Théorie des réseaux*. https://fr.wikipedia.org/wiki/Théorie_des_réseaux. Consulté le 8 janvier 2019. Nov. 2018.

[6] Christophe Coupé. *De l'origine du langage à l'origine des langues : Modélisations de l'émergence et de l'évolution des systèmes linguistiques*. École doctorale de Sciences Cognitives. Jan. 2003.

[7] Merritt Ruhlen. *Tracing the evolution of the mother tongue*. ISBN 0-471-58426-6. 1994.

Naissance d'un langage

Résumé

Soit un ensemble de villages, possédant chacun un vocabulaire limité et pouvant communiquer entre eux. À tour de rôle, les villages tentent d'émettre un mot de leur vocabulaire à leurs voisins, qui, s'ils le reçoivent, peuvent ou non adopter son usage. On s'intéresse à l'évolution du vocabulaire des différents villages au cours du temps et à l'influence des différentes paramètres (nombre de villages, complexité du voisinage, taille initiale de vocabulaire, etc.) sur la simulation.

Mots-clés

Simulation, Simulation à événements discrets, Java, JavaFX, Théorie des graphes

Abstract

Let a set of villages, each with its limited own vocabulary and all able to communicate with each other. One by one, the villages try to send a word of their vocabulary to their neighbors, who, if they receive it, may or may not adopt its use. We are interested in the evolution of these villages over time and the impact of the main parameters (number of villages, neighborhood complexity, initial vocabulary size, etc.) on the simulation.

Keywords

Simulation, Events driven simulation, Java, JavaFX, Graphe theory

Tuteur académique
Christophe Lenté

Étudiants
Charles MECHERIKI (DI4)
Yongda LIN (DI4)