Chapitre 3

Modèles, simulations et outils informatiques

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.
John von Neumann.

L'informatique et la modélisation (principalement issue à l'heure actuelle de l'intelligence artificielle) constituent un champ en plein essor pour l'étude de l'origine du langage, et de l'évolutions des langues dans une moindre mesure. Un bon indicateur de cette tendance est le nombre croissant de "modélisateurs" qui présentent leurs travaux dans le cadre de conférences sur l'évolution du langage.

Face à des questions aux réponses parfois peu intuitives, un modèle et son implémentation informatique représentent souvent un champ d'investigation fructueux. Ils permettent par exemple d'identifier les raisons minimales permettant d'expliquer un phénomène réel, et de simplifier ainsi l'enchaînement des causes et des effets. Les puissances de calcul sans cesse croissantes permettent de simuler des situations appartenant au passé ou échappant à l'expérimentation physique.

Après un très rapide historique de l'évolution des techniques informatiques, nous tenterons de définir ce qu'est un modèle, avant de définir quelques grands paradigmes utilisés pour la recherche sur l'origine du langage, et présenter rapidement les recherches qui ont été menées jusqu'à aujourd'hui. Dans une seconde partie, nous présenterons les outils logiciels que nous avons développés pour nos simulations.

3.1 Panorama des recherches informatiques sur l'origine et l'évolution du langage et des langues

3.1.1 Une brève histoire de l'informatique

De la machine de Neumann aux super-calculateurs neuronaux

L'informatique trouve ses racines dans les premières machines mécaniques des XVIIème et XVIIIème siècle : la Pascaline de Pascal en 1643, qui réalisait l'addition et la soustraction, la machine de Leibniz en 1673, qui réalisait elle les 4 opérations de base, le canard mécanique de Vaucanson en 1738, la machine de Babbage en 1833, réalisée initialement pour le calcul de tables numériques pour la navigation et qui intégrait nombres des découvertes précédentes...

Le XIXème siècle verra l'émergence d'une logique dégagée de la philosophie et orientée vers les mathématiques, avec notamment les travaux de Boole (fondateur de l'algèbre de Boole), Frege, Russel ou Hilbert. Cette logique servira de base à l'informatique théorique du XXème siècle et aux travaux des logiciens de ce même siècle.

A la suite des premières réalisations purement mécaniques, l'invention de l'électricité permit l'apparition des premières machines électro-mécaniques, puis des machines électroniques suite à l'invention du transistor (en 1947 par John Bardeen, Walter Brattain et William Shockley des laboratoires Bell) et des circuits intégrés (en 1959 chez Texas Instruments).

D'un point de vue technique, l'informatique va surtout se développer pendant la seconde guerre mondiale, avec les besoins de l'armée américaine, en particulier pour des questions de cryptographie (encodage et décryptage de messages secrets), de calculs de tables de tirs ou de localisation de sous-marins. Elle va aussi servir au développement de la première bombe A. Une des premières simulations sera d'ailleurs celle du calcul de la hauteur optimale pour l'explosion de la bombe.

Sur un plan théorique, le milieu du XXème siècle va consacrer la naissance de l'informatique théorique, avec les travaux de grands mathématiciens, logiciens et ingénieurs comme Kurt Gödel (1906-1978), Alan Turing (1912-1954) (qui a joué un rôle considérable dans la découverte des codes secrets allemands) et John Von Neumann (1903-1957). Le second, en introduisant le concept de machine universelle (la machine de Turing), établit les limites des machines sur les plans philosophique et calculatoire. Von Neumann établit quant à lui l'architecture qui porte son nom, et qui est toujours celle de la très grande majorité des ordinateurs d'aujourd'hui (voir la figure 3.1).

A la fin des années 1940, on assiste également au développement de la cybernétique, initiée par l'américain Norbert Wiener (1894-1964) en 1948, ainsi que de la théorie de l'information, dont les bases sont dues à Claude Elwood Shannon (1916-2001).

D'une façon générale, la seconde partie du XXème siècle verra ensuite l'accroissement de la puissance des machines parallèlement à leur miniaturisation, et le développement de systèmes d'exploitation (Unix, IBM (34, 38...), AS-400, Windows, Linux...) et de logiciels toujours plus performants. Un ordinateur de plusieurs tonnes, qui occupait une pièce entière il y a 50 ans, et aujourd'hui remplacé par un portable de moins de deux kilogrammes...

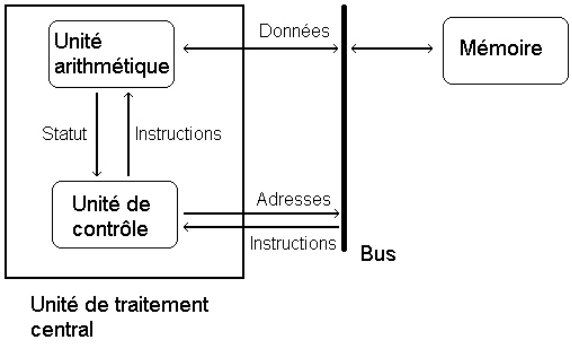

FIG. 3.1 – L'architecture de Von Neumann

Tendances évolutives de l'informatique

La tendance générale de l'informatique depuis plusieurs décennies est une envolée vers toujours plus de puissance : plus de puissance de traitement, mais aussi de mémoire, de capacités de stockage ou encore de techniques graphiques... La loi exponentielle qui régit cette progression n'est toujours pas contredite à l'heure actuelle, malgré l'approche des limites de miniaturisation des composants (à une échelle inférieure, les lois de la mécanique quantique viennent troubler le bon fonctionnement de la machine). Les nouveaux composants quantiques ou optiques laissent cependant envisager des gains en puissance toujours plus importants pour le futur.

Cette progression a bien sûr permis et continuera à permettre des simulations toujours plus lourdes. Si une simulation n'est pas toujours "gourmande" en ressources, la répétition d'un grand nombre d'expériences (pour étudier des jeux de paramètres par exemple, ou étudier des phénomènes stochastiques) ou certains phénomènes complexes (simuler un grand nombre de locuteurs par exemple) nécessitent une puissance importante pour ne pas nécessiter des jours ou des mois de traitement.

Parallèlement à cette tendance, on peut aussi noter le développement d'analogies avec la nature et le réel. Ces analogies peuvent se rencontrer au niveau des techniques de modélisation, mais également au niveau du matériel, avec le développement de nouvelles architectures, comme celles à base de processeurs massivement parallèles.

3.1.2 Modélisation et simulations informatiques

Qu'est-ce qu'un modèle ?

Nous pouvons définir un modèle comme une abstraction d'un phénomène qui permet, à travers une simplification du réel, d'appréhender les mécanismes de ce phénomène. Il s'agit en effet de parvenir à éliminer les éléments non pertinents vis à vis du phénomène (le "bruit", la variation aléatoire de certaines grandeurs) pour mieux se consacrer aux paramètres significatifs.

La modélisation d'un phénomène repose sur une "technique de modélisation" qui peut emprunter des formes très variées, selon les caractéristiques du phénomène étudié ou les présupposés du modélisateur. De très nombreuses approches cohabitent pour la modélisation de phénomènes biologiques, physiques, linguistiques... Nous pouvons citer pour exemple une approche mathé-

matique très répandue dans de nombreux domaines, les équations différentielles, dans la grande variété qui les caractérise [Murray, 1984] :

- en physique : développement des tâches des robes de nombreux animaux, écoulements d'air ou de fluides ;
- en chimie : évolutions des concentrations lors d'une réaction chimique, cinétiques de réactions (catalyseurs, inhibiteurs...) ;
- en biologie : évolution des populations, diffusion d'épidémies ;
- en linguistique : modélisation du phénomène de diffusion lexicale [Wang, 2002a].

Selon la situation à modéliser, les équations différentielles ne sont bien sûr pas forcément judicieuses. Leur emploi suppose ainsi des phénomènes suffisamment réguliers que l'on puisse représenter par des variables (pseudo-)continues.

Notons que la notion de modèle semble proche de celle de théorie, mais à la différence du modèle, il nous semble que la théorie ne s'appuie pas nécessairement sur une réduction du phénomène réel qu'elle étudie.

Rôle de la simulation

Une somme d'expériences ne peut jamais me prouver que j'ai raison ;
une seule expérience peut n'importe quand me prouver que je me suis trompé.
Albert Einstein.

Construire un modèle et en disposer est une première chose, tester sa validité en est une autre. Dans certaines situation, ce test peut s'effectuer en comparant les prédictions du modèle à la réalité. Un exemple très célèbre est la vérification de la théorie de la relativité générale d'Einstein lors d'une l'éclipse totale de soleil en 1916 : l'observation d'étoiles à proximité du soleil masqué par la lune confirme la courbure des rayons lumineux par la masse solaire. Toujours dans le cadre de la relativité générale, l'application du modèle aux mesures du périhélie de Mercure permet de corriger les erreurs minimes de calculs qui persistaient avec le modèle newtonien, et de valider ainsi partiellement le nouveau modèle.

Il existe toutefois des cas où il n'est pas possible de tester directement un modèle par simple observation des faits[25]. L'expérimentation peut alors remédier au problème, et la simulation informatique fait partie des expérimentations possibles. De nombreux modèles sont trop complexes pour qu'on puisse en concevoir immédiatement les effets. Nous pouvons prendre l'exemple de la modélisation de la circulation de Los Angeles : il est possible d'élaborer un modèle de la circulation, mais impossible d'envisager les prédictions de ce modèle sans une simulation informatique, si le nombre de véhicules pris en compte est un tant soit peu important. Des simulations sont donc effectuées, qui permettent de visualiser les résultats du modèle, et d'envisager des solutions pour fluidifier la circulation.

[25] Il est à noter que la modélisation informatique peut parfois aussi remplacer avantageusement la réalisation d'expériences réelles : explosions nucléaires, "crash-tests" automobiles...

Daniel Nettle propose une vue légèrement différente de l'intérêt des simulations informatiques. Plus que de vérifier un modèle, il s'agit ici également de participer à son élaboration. Les simulations servent à éliminer les "ingrédients" du modèle qui ne joueraient pas un rôle indispensable dans l'engendrement du phénomène observé. Il peut en effet être difficile de détecter *a priori* de tels éléments si la complexité du phénomène est importante. La démarche est ici constructive :

"A computer simulation is never a realistic replication of the situation it aims to elucidate. Thus it can very rarely be used to make precise empirical predictions about real world processes. One might therefore ask what the point is of doing it. The answer is that though it is not a total replication of a real world situation, it attempts to take to take the main relationships of a real world situation and explore their general effects across a range of conditions. This is useful because complex iterated processes of the kind found in social evolution can have unlooked-for, complex dynamics which cannot be predicted by simple thought or deduction. A simulation, however rudimentary, is thus an improvement over a merely verbal argument, in deciding what general conditions must obtain for languages to evolve in the way that they do. It is in this spirit that I would like the following simulation to be taken ; highly simplistic, but, I hope, interesting as a way of exploring issues surrounding language change which is more systematic and better supported than mere speculation.

Critics of computer simulations often complain that the author has simply built into the model whatever he desires, so when that desideratum results it is not interesting. This is a misunderstanding of what simulations such as this one are for. I will indeed tweak this system until it produces results that resemble real linguistic change in some ways. The interest lies therefore not in what it can be made to do, but rather in what assumptions and initial conditions have to be included to make it behave in the desired way. These assumptions and conditions may give us some general insights into the conditions of real language change."[Nettle, 1999c] (p. 103)

La démarche adoptée par Nettle nous semble la plus judicieuse qui soit. Elle ne prend toutefois pas vraiment en compte, ou ne la mentionne pas explicitement, la possibilité d'observer parfois des événements qui étaient imprévisibles à cause d'une grande complexité computationnelle. Observer des phénomènes imprévus permet souvent de soulever de nouvelles questions, en s'interrogeant sur l'existence du phénomène dans la réalité et sur ses conséquences. La démarche se fait ici plus exploratoire, dans le sens où le but du modèle est alors de mettre à jour des phénomènes complexes à partir d'un jeu de conditions particulier, plutôt que de découvrir les conditions minimales nécessaires à l'existence d'un phénomène donné *a priori*.

Comme le souligne Nettle, une simulation, même rudimentaire, est toujours plus forte qu'un argument verbal. Même dans des situations où les données de la discipline concernée ont abouti à un consensus clair et des théories bien acceptées, des simulations peuvent être utiles pour confirmer ces dernières et éventuellement découvrir des difficultés non suspectées. Toute théorie s'appuie toujours sur un certain nombre de présupposés, et comme dans une démonstration mathématique, les points les plus difficiles sont souvent ceux qui sont considérés comme triviaux. . .. Un examen approfondi de situations complexes peut ainsi mettre au jour des impossibilités et entraîner une réforme des théories. Ce scénario peut être envisagé par le biais de simulations informatiques, mais également à l'aide d'autres outils, et les progrès scientifiques sont le plus souvent accomplis sur la base d'une remise en question des théories bien établies [Kuhn, 1983].

Mathématisation et modélisation

Toute la nature n'est que mathématique.
Galilée.

Nous avons introduit dans la section précédente le modèle mathématique des équations différentielles comme outil possible de modélisation d'un phénomène réel. Il est intéressant de constater que nombre de phénomènes réels ne peuvent être modélisés sous une forme mathématique (ceci n'est d'ailleurs souvent pas le but des auteurs de modèles). D'autres peuvent parfois être représentés à un certain niveau par un système mathématique, mais celui-ci est souvent trop complexe pour pouvoir être résolu par une méthode analytique. Les frontières de résolution exacte de systèmes comme ceux d'équations différentielles sont en fait vite atteintes, et la simulation numérique seule permet une résolution approchée du système. Il est à noter que la simplicité du modèle mathématique ou la possibilité même d'utiliser un modèle mathématique dépend bien sûr du niveau de détail avec lequel se fait la description du phénomène réel dans le modèle, en particulier selon les approximations qui y sont effectuées. On pourra se référer à [Popescu-Belis, 2002] pour juger de ces notions dans le cadre de modèles mathématiques liés à l'émergence de conventions lexicales dans une population d'agents.

Nous venons de présenter quelques points relatifs à la modélisation et à la simulation. Comment s'inscrivent les questions de l'origine et de l'évolution du langage et des langues dans ce contexte ? Parmi les nombreux modèles qui peuvent être échafaudés autour de cette question, un certain nombre se rapportent à des spécificités que nous avons déjà évoquées dans les premier et second chapitres : absence de fossiles linguistiques, dynamique complexe de populations de locuteurs avec une certaine variabilité inter-individuelle... Des paradigmes particuliers et des simulations, que nous allons maintenant détailler, peuvent mettre au jour des éléments nouveaux difficiles à découvrir par d'autres approches.

3.1.3 Principaux paradigmes de modélisation

Afin de tenter de faire le tri entre différentes approches, nous pouvons mentionner trois caractérisations du langage qui représentent assez bien les conceptions des modélisateurs, et qui se rapprochent des trois dimensions du langage développées au chapitre 2 :

- le langage est un système distribué dans une population de locuteurs. Il s'agit avant tout d'un outil d'échange d'information, dont les caractéristiques reposent sur les interactions entre individus ;
- le langage est un système dynamique adaptatif complexe. Il est composé de différents niveaux et structures qui évoluent de façon entrelacée et complexe, en s'adaptant aux besoins de communication ;
- le langage est un outil ou système **cognitif**, transmis de génération en génération grâce à des phases d'acquisition.

Ces définitions même appellent différentes questions, qui constituent la base du travail des modélisateurs :

- comment des interactions diadiques entre individus peuvent-elles conduire à l'émergence de conventions partagées par toute une communauté d'individus ?
- quel est l'impact des structures sociales, physiologiques et psychologiques des individus sur la structure du langage qu'ils utilisent, que ce soit au niveau lexical, phonologique, syntaxique... ?
- comment l'acquisition de la syntaxe est-elle possible dans un cadre théorique particulier (théorie des principes et des paramètres, systèmes cognitifs non dédiés au langage...) ?
- quels sont les facteurs d'évolution du langage et des langues ?

Afin d'aborder ces questions, différents paradigmes de modélisations, plus ou moins mathématiques, offrent des propriétés intéressantes et "miment" d'une certaine façon les phénomènes réels. Les paragraphes suivants détaillent les plus couramment utilisés.

Les systèmes multi-agents

La notion d'agent est née à la fin des années 1980, dans le prolongement de la notion d'objet que nous décrirons par la suite. Un agent se définit comme une entité autonome, dotée de propriétés, et capable d'interagir avec son environnement, et en particulier avec d'autres agents, éventuellement de même nature.

Comme son nom l'indique, un système multi-agents (SMA) est un ensemble d'agents. La capacité d'interaction qui les caractérise va pouvoir être utilisée à plusieurs fins, par exemple pour résoudre un problème donné de façon décentralisée (on parle d'*intelligence artificielle distribuée* ou *IAD*).

Les deux définitions précédentes se situent à un niveau très conceptuel, et laissent par conséquent une grande liberté dans la définition de l'agent dans un contexte donné. Dans le cadre de la modélisation, le degré de complexité des concepts, phénomènes ou organisations vivantes qui sont retenus pour la "traduction" en agents peut ainsi être adapté en fonction de l'étude. En prenant l'exemple de modélisations des organisations sociales, on pourra ainsi choisir de se placer au niveau des individus qui les composent si l'on s'intéresse à l'impact de leurs relations sur la structure de l'organisation. Si l'on se penche plutôt sur les relations entre ces organisations, alors il pourra être plus judicieux de choisir ces organisations comme agents, munis de propriétés éventuellement complexes traduisant l'ensemble des relations des individus à l'intérieur de ces structures[26]. Nous reproduirons ces deux démarches au chapitre 6.

Un champ d'application très développé des systèmes multi-agents est celui de la modélisation des phénomènes sociaux, comme par exemple les mécanismes financiers, en particulier dans le cadre de la rationalité limitée des acteurs [Edmonds, 1999]. Il s'agit d'une part de comprendre les caractéristiques du vivant ou des sociétés humaines, en mettant en lumière les mécanismes sous-jacents qui rendent compte de la complexité des phénomènes observés ; d'autre part, éventuellement, d'appliquer ces mécanismes pour résoudre des problèmes dans des domaines différents.

[26] D'une façon générale, une relation s'établit entre le niveau de *granularité* des SMA (1 agent pour un concept général ou au contraire plus minimal) et la complexité des agents de ce système. Une granularité assez faible (un agent pour un concept "haut-niveau") pourra être contrebalancée par une complexité assez élevée des agents. Au contraire, on pourra définir des agents minimaux et extrêmement simples, comme le sont par exemple les agents pûrement réactifs, qui ne font que réagir de façon "quasi-automatique" aux modifications de leur environnement (par exemple des fourmis artificielles qui se dirigent vers les sources de phéromones les plus intenses dans leur espace perceptif [Bonabeau and Theraulaz, 2000]). Le choix du niveau de granularité traduit aussi parfois les contraintes en terme de puissance des machines.

Un exemple est celui de l'étude computationnelle des comportements d'insectes sociaux comme les fourmis, qui après avoir fourni des mécanismes explicatifs pour les études éthologiques, a produit des applications très pertinentes dans la gestion du trafic des réseaux comme Internet [Schoonderwoerd et al., 1996].

Les modélisations portant sur l'origine du langage se situent évidemment dans ce dernier cadre. L'isomorphisme entre les agents et les locuteurs est très naturel, et les systèmes multi-agents traduisent fort bien l'aspect distribué du langage dans une communauté de locuteurs.

Certains auteurs, comme Marvin Minsky, ont également proposé d'appliquer ce concept à la cognition dans son ensemble, et ont proposé le terme de société de l'esprit [Minsky, 1987].

Les réseaux de neurones artificiels

Il est possible de représenter la cognition d'un locuteur de façon plus ou moins complexe. Afin de modéliser les capacités d'apprentissage et de généralisation d'un locuteur, une des possibilités est le recours aux réseaux de neurones artificiels.

Ces réseaux, d'inspiration biologique, ont été introduit en 1943 par Mac Culloch et Pitts, qui les premiers proposèrent une abstraction du neurone biologique sous la forme d'un automate à deux états calculant sa sortie booléenne à partir de valeurs d'entrée et d'un seuil d'activation. L'analogie avec le neurone réel, recevant des influx nerveux par ses dendrites et déchargeant en fonction de cette entrée est immédiate, même si la réduction opérée par le modèle est très importante. La mise en commun de neurones artificiels connectés entre eux conduit au concept de réseaux de neurones.

Cette définition, comme pour les systèmes multi-agents, laisse une grande liberté tant pour la topologie des réseaux que pour la fonction d'activation des neurones. De fait, différents types de réseaux existent aujourd'hui, aux propriétés architecturales et fonctionnelles distinctes :

- le caractère supervisé ou non (voir mixte) de l'apprentissage. Dans le premier cas, il est nécessaire pour l'apprentissage de présenter au réseau la sortie qu'il doit faire correspondre à une entrée. Les réseaux non supervisés règlent quant à eux les poids de leurs connexions selon des processus "autonomes", souvent basés sur des principes d'auto-organisation ;
- la typologie de type "feed-forward" ou récurrente. Les réseaux du premier type voient une propagation de l'activation des neurones depuis une couche d'entrée vers une couche de sortie (entre elles se trouvent zéro, une ou plusieurs couches intermédiaires ou *cachées*). Les suivants possèdent au contraire des connexions dans les deux sens, conduisant à des boucles de (rétro-)propagation de l'activité ;
- l'apprentissage et le calcul d'une fonction associant des valeurs de la couche d'entrée à des valeurs de la couche de sortie, ou la construction d'un ensemble de prototypes à partir d'un ensemble d'éléments. Dans le premier cas, suite à une période d'apprentissage, de nouvelles entrées peuvent être présentées, et le réseau calcule la valeur de la fonction apprise pour celles-ci. Dans le second cas, de nouveaux éléments pourront être rapprochés des prototypes dégagés de l'ensemble d'apprentissage ;
- le recours à des lois d'inspiration biologique, comme la loi de Hebb, postulée par Donald Olding Hebb (1904-1985) en 1949 :
  "When an axon of cell A is near enough to excite B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" [Hebb, 1949] (p. 62) ;

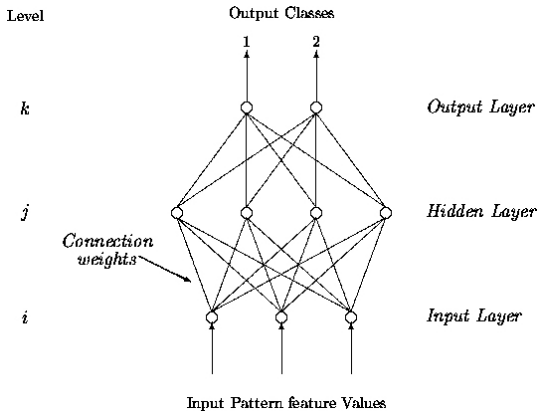

FIG. 3.2 – Schéma d'un perceptron multi-couches

- l'utilisation de processus stochastiques...

Il existe de très nombreux réseaux différents, qui empruntent et combinent les propriétés précédentes. Nous pouvons citer les suivants :

- le perceptron multi-couches ou PMC, réseau de type feed-forward (voir figure 3.2), à apprentissage supervisé, pour l'apprentissage de fonctions. La technique d'apprentissage, baptisée *rétro-propagation de gradient* fut à l'origine du renouveau du courant connexionniste dans les années 1980, suite à une désaffection liée aux capacités limitées des premiers réseaux, et le développement de l'intelligence artificielle symbolique ;
- la carte de Kohonen, due à Teuvo Kohonen [Kohonen, 1990], pour l'extraction de prototypes, non-supervisée. Bien que ce réseau ait été conçu initialement comme un outil d'ingénierie, certaines de ses propriétés se retrouvent au niveau de certaines régions cérébrales corticales [Georgopoulos et al., 1988] ;
- les réseaux de Hopfield, la machine de Boltzmann (utilisant des techniques stochastiques), les réseaux récurrents, ceux gérant les signaux temporels...

Les systèmes d'apprentissage de "haut niveau"

A l'inverse des réseaux de neurones que l'on peut qualifier de systèmes d'apprentissage *subsymboliques* (puisqu'ils ne manipulent que des valeurs numériques et non des représentations plus abstraites), il est possible d'équiper les locuteurs artificiels de systèmes d'apprentissage, de perception et de production linguistiques basés sur des lois et des éléments de haut niveau. Plusieurs exemples de simulations permettront de mieux envisager cette approche dans quelques paragraphes.

Les algorithmes génétiques

Nous pouvons citer pour finir notre rapide tour d'horizon la technique des *algorithmes génétiques*. Développée par John Holland [Holland, 1975], et surtout utilisée initialement pour des problèmes d'optimisation en ingénierie, elle repose sur une analogie forte avec les mécanismes biologiques de la reproduction sexuée. Le mécanisme de ces algorithmes est en effet le suivant :

1. choix d'une population d'éléments initiaux que l'on peut dénommer chromosomes, dotés de différentes caractéristiques ;
2. évaluation de la qualité de chaque élément à l'aide d'une fonction d'évaluation ou "fonction de fitness" ;
3. sélection des chromosomes les plus performants et croisement (crossing-over) de ces chromosomes, accompagné de mutations aléatoires de leurs caractéristiques ;
4. retour à l'étape 2 avec la nouvelle population constituée des chromosomes créés à l'étape 3.

De même que la sélection naturelle sélectionne les caractéristiques qui favorisent le succès reproductif d'un individu, les algorithmes génétiques mettent au jour le jeu de caractéristiques qui répondent le mieux à l'évaluation de la fonction de fitness.

Les algorithmes génétiques peuvent être utilisés pour l'apprentissage de réseaux de neurones artificiels comme les perceptrons multi-couches (à la place de l'algorithme de rétro-propagation). Ils sont surtout utilisés dans les simulations qui nous concernent pour reproduire des mécanismes évolutifs plausibles.

3.1.4 Principaux courants de recherche

Emergence des conventions

Initiée principalement par Luc Steels et Frédéric Kaplan, l'étude de l'émergence de conventions linguistiques s'attaque à la question de l'apparition d'un code partagé dans une population d'agents uniquement sur la base d'interactions répétées entre deux agents. Ce code peut s'établir à différents niveaux.

Au niveau lexical. L'émergence de conventions lexicales ne prend pas en compte l'évolution de la structure interne des mots, mais étudie les possibilités d'apparition d'un système de dénomination pour les éléments d'un univers sémantique. Plus concrètement, des agents évoluent dans un univers peuplé de différents objets ou concepts. Chaque agent peut accéder à ces concepts, qui sont partagés et accessibles à tous. De plus, chacun dispose d'une mémoire capable de retenir des mots qu'il peut associer aux différents concepts de son environnement. Les agents peuvent entrer en interaction deux à deux pour tenter d'adopter les mêmes mots pour les différents objets. La question est de savoir si un système de mots cohérent pour tous les agents peut émerger sur la base d'interactions diadiques. La figure 3.3 présente le déroulement d'un schéma interactif possible, avec les différentes possibilités d'évolution. De nombreuses autres stratégies voisines peuvent être envisagées (voir par exemple [Steels, 1997a] (p. 13) ou [Ke et al., 2002a]), qui conduisent aux mêmes résultats : après une période plus ou moins longue (fonction du nombre d'agents, du nombre de concepts...), les agents finissent par converger vers un système partagé de mots pour décrire les différents objets. La convention est résistante à un renouvellement des agents, à condition que celui-ci ne soit pas trop important (la limite est appelé *flux limite de résilience*) [Kaplan, 2000] (p. 71-75).

L'influence des homophones peut être étudiée en limitant l'espace des éléments permettant de constituer de nouveaux mots : par exemple, si les mots peuvent être composés par combinaison de 3 lettres choisies au hasard dans un ensemble de 4, il n'existera que 64 mots possibles.

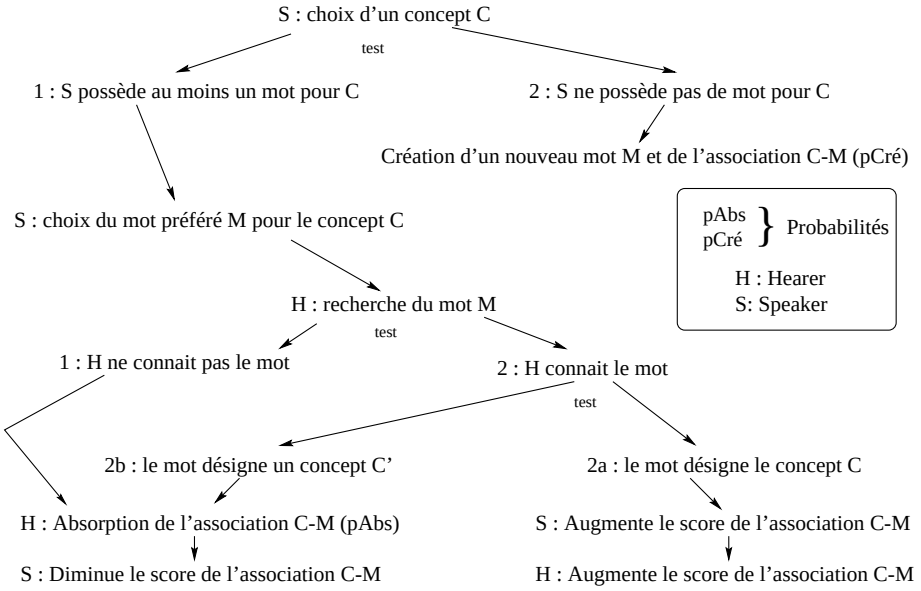

FIG. 3.3 – Interaction lors d'un naming game

Au hasard des créations de mots, des homophones peuvent émerger qui viennent gêner la communication. De même, l'invention de nouveaux mots par des agents différents pour un même concept conduit à l'existence de synonymes.

De nombreux raffinements peuvent être apportés au modèle, qui permettent d'étudier différents aspects du cas général. Un premier exemple est l'ajout d'une dimension spatiale, qui vient introduire une hétérogénéité dans les communications entre agents : plus deux agents sont proches, plus la fréquence de leurs interactions est importante [Steels, 1996]. En formant des clusters d'individus, des systèmes de dénomination propres à chaque cluster émergent, tandis qu'une *"lingua franca"* s'instaure entre les différents clusters si les interactions entre eux sont suffisantes.

Une autre direction d'étude est celle des conditions de contacts. Steels étudie l'évolution du lexique des agents de trois clusters initialement séparés (avec un nombre d'interactions très réduit entre eux), puis mis en contact (accroissement du nombre d'interactions inter-clusters). Les résultats de ces expériences mettent en évidence une hausse du bilinguisme chez les agents, ainsi qu'un mélange graduel des différentes "langues" initiales pour aboutir à une situation de convergence parfaite [Steels, 1997a].

Avec Egidio Marsico et François Pellegrino, nous avons prolongé le cadre précédent afin d'étudier l'impact des tailles des populations mises en contact, ainsi que l'importance du prestige des agents mis en contacts [Marsico et al., 2000]. L'idée était que des agents plus prestigieux imposaient plus facilement les mots de leur lexique lors du contact. Renforçant ce phénomène ou le minimisant, une plus petite taille de population rendait plus difficile la préservation d'un lexique après le contact. Ces situations sont importantes si l'on songe aux situations de la préhistoire, où les populations humaines étaient distribuées sous forme de groupes de quelques dizaines d'individus (voir chapitre 4). Les résultats montrent qu'au delà d'un certain ratio des tailles de populations, le lexique de la plus petite est totalement remplacé lors du contact linguistique.

Même un écart de prestige important ne peut alors compenser ce déséquilibre. Les courbes des figures 3.4, 3.5, 3.6 et 3.7 montrent l'évolution des cohérences des deux populations, ainsi que deux indicateurs de l'emprunt lexical pour deux situations de contact : la première figure illustre la situation de deux populations de même taille (30 individus) et même prestige, alors que la seconde met en jeu deux populations de tailles différentes (30 agents pour la population bleue, 40 pour la population rouge). La cohérence est une grandeur numérique entre 0 et 1 qui indique le degré d'agrément des agents sur les mêmes mots pour désigner les (10) objets de leur environnement. En ce qui concerne l'emprunt, les deux courbes (couleur foncée et couleur claire) concernent respectivement le nombre de mots étrangers (venus de la seconde population) connus en moyenne par un locuteur, et le nombre moyen de mots étrangers utilisés de façon préférentielle par un agent.

Si l'on examine les courbes du graphe de la figure 3.4, on observe qu'avant contact (à gauche de la barre verticale), les cohérences des deux populations augmentent jusqu'à atteindre une valeur proche de 1, ce qui signifie que les agents de chaque population atteignent un agrément sur les mots à employer. Peu de temps après le contact, on constate une baisse de la cohérence identique pour les deux populations. Cette baisse traduit une désorganisation des vocabulaires des agents à cause de l'introduction de nouveaux mots dans les interactions. Cependant, on remarque que les cohérences finissent par remonter lentement : les agents des deux populations établissent une nouvelle convention sur un ensemble de mots qui emprunte aux vocabulaires initiaux des deux populations. Les courbes du graphe de la figure 3.5 traduisent ce phénomène : très peu de temps après le contact, on constate que tous les agents connaissent un mot étranger pour désigner l'ensemble des 10 objets de leur environnement. En ce qui concerne le choix des mots qui formeront le vocabulaire final, la proximité des courbes d'emploi préférentiel de mot étranger montre que ce vocabulaire est composé à 50% de mots issus du lexique de la première population, et à 50% de mots issus de celui de la seconde.

Si l'on regarde maintenant les courbes de la figure 3.6 où la situation est asymétrique entre les deux populations, on constate que la plus petite subit de façon beaucoup plus importante le contact linguistique. La cohérence de cette population baisse en effet beaucoup plus fortement que celle de la population de plus grande taille. Celle-ci voit son vocabulaire peu déstructuré, et celui-ci constituera d'ailleurs la majeure partie du vocabulaire après contact et stabilisation. Les courbes du graphe de la figure 3.7 traduisent cet état de fait, et le faible nombre de mots issus de la petite population utilisés de façon préférentielle par les membres de la population de grande taille.

En association avec Jinyun Ke et Feng Wang du laboratoire LEL de la City University of Hong Kong, nous avons également étudié la façon dont les homophones peuvent persister dans un modèle où des interactions existent entre les concepts [Ke et al., 2002b]. L'utilisation dans une interaction des relations sémantiques qu'entretiennent les composants d'une phrase (comme par exemple dans la phrase *"Le chien aboie."*) permet en cas d'ambiguïtés causées par de l'homophonie de faire le plus souvent le choix correct parmi les différents homophones en compétition.

Un espace bidimensionnel torique permet de représenter les sens des objets du monde : chaque sens est identifié par deux coordonnées dans cet espace. Il est possible dès lors d'établir une distance sémantique entre les mots. L'interaction est toujours basée sur le principe du naming game, mais les agents échangent cette fois deux mots. Le nombre limité d'éléments "phonémiques" qui servent à construire les mots conduit à un nombre lui aussi limité de mots distincts. Si le nombre

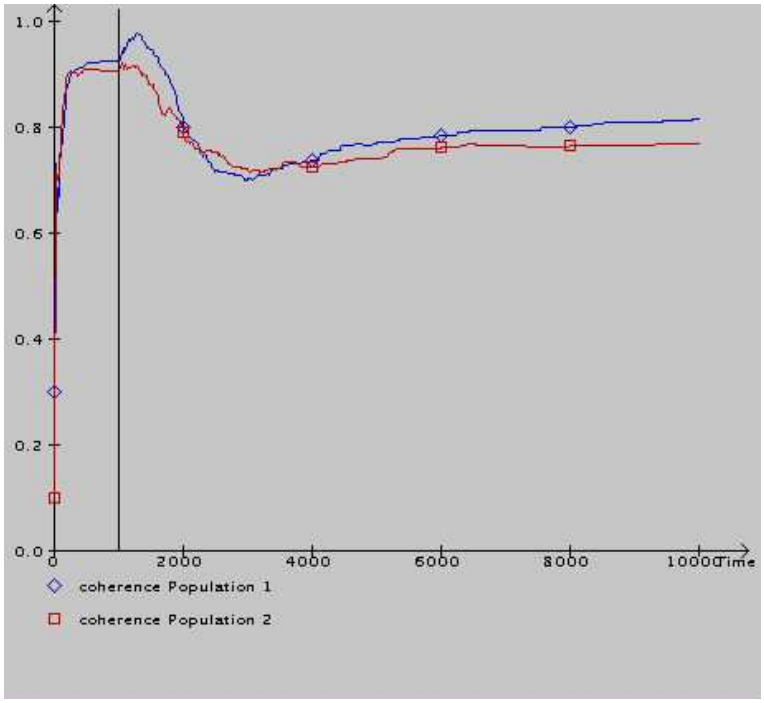

FIG. 3.4 – Evolution de la cohérence de deux populations de taille identique en situation de contact linguistique

de concepts à exprimer dépasse cette limite, des homophones existent inévitablement (et même en dessous de cette limite, puisque les mots sont composés aléatoirement). Lors d'une homophonie au cours d'une interaction, le récepteur envisage l'ensemble des combinaisons possibles entre les mots de l'interaction, et sélectionne le couple de plus faible distance sémantique. En outre, un procédé d'élagage (*"pruning"*) permet d'éliminer si besoin les mots qui présentent un succès de communication trop faible (après plusieurs interactions).

Ce procédé permet de surmonter l'homophonie de façon très significative, et des ratios moyens de 3 ou 4 concepts pour une même forme phonémique (un mot) permettent néanmoins un succès de communication important. Les courbes noires des graphes des figures 3.8, 3.9, 3.10 et 3.11 présentent respectivement les situations suivantes : nombre de concepts égal au nombre de mots distincts possibles et interactions à 1 mot, nombre de concepts égal au nombre de mots distincts possibles et interactions à 2 mots, nombre de concepts égal à 3 fois le nombre de mots distincts possibles et interactions à 1 mot et enfin nombre de concepts égal à 3 fois le nombre de mots distincts possibles et interactions à 2 mots. Dans les deux premiers cas, le succès de communication après émergence du lexique est proche de 1. En particulier, dans le premier cas, l'élagage permet à la communauté d'agents de finir par découvrir l'ensemble des mots distincts, et à éliminer l'homophonie du système comme cela est possible.

Dans les deux cas suivants, l'homophonie est très importante. L'absence de contexte sémantique ne permet pas de surmonter ce problème (figure 3.10), et le succès de communication est proche de 33%, ce qui correspond au pourcentage de chance de trouver le mot correct parmi trois homophones. Avec l'aide du contexte, l'homophonie peut-être surmontée, et le succès de communication peut croître (90% sur le graphe).

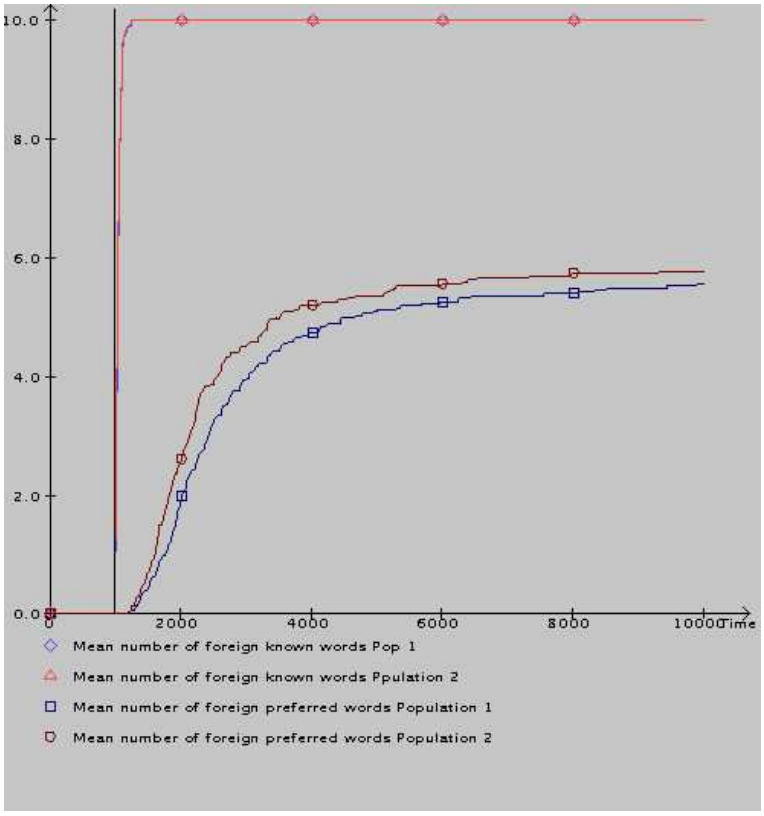

FIG. 3.5 – Evolution du vocabulaire emprunté de deux populations de taille identique en situation de contact linguistique

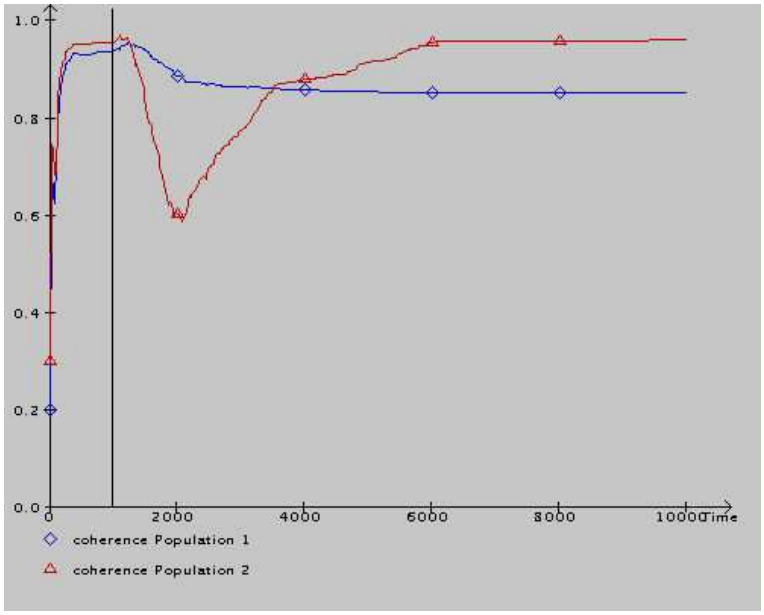

FIG. 3.6 – Evolution de la cohérence de deux populations de tailles différentes en situation de contact linguistique

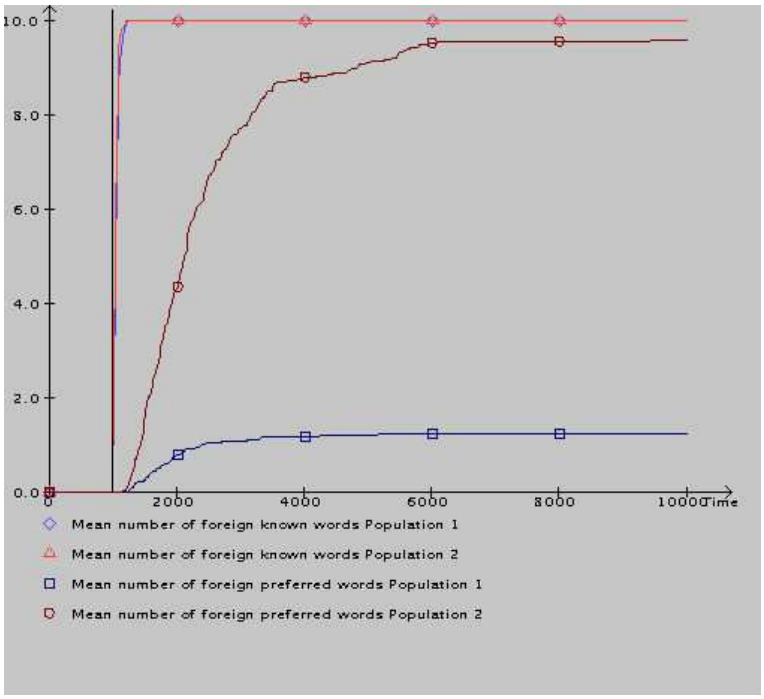

FIG. 3.7 – Evolution du vocabulaire emprunté de deux populations de tailles différentes en situation de contact linguistique

Catégorisation sémantique. L'existence *ad hoc* de concepts est un présupposé important, et Steels a également étudié la façon dont une catégorisation partagée d'un univers sémantique peut émerger dans une population d'agents.

[Steels, 1997b] résume tout d'abord comment une catégorisation progressive d'un espace continu peut émerger chez un seul agent. Le jeu utilisé ici est un jeu de discrimination, baptisé *"discrimination game"*. Le but est de parvenir à distinguer un objet ou une situation parmi un ensemble d'autres. Les objets sont identifiables d'emblée, et sont perçus via un ensemble de senseurs qui peuvent acquérir les valeurs d'un ensemble de caractéristiques de ces éléments. Ces détecteurs ont en outre la capacité de fournir des valeurs discrètes à partir d'une plage continue de valeurs. Pour un épisode donné où un agent cherche à distinguer deux objets o_1 et o_2, l'agent utilise ces capteurs pour extraire deux ensembles de traits discrets relatifs aux deux objets. Si ces deux ensembles sont distincts, le jeu est réussi. Dans le cas contraire, l'agent peut soit raffiner l'une de ses segmentations pour une plage continue de valeurs, soit créer un nouveau senseur pour une caractéristique de l'objet encore non exploitée. Après un certain nombre d'épisodes, une segmentation particulière émerge, qui correspond à un découpage des plages de valeurs continues qui permet (mais pas forcément de façon optimale) la discrimination de l'ensemble des objets.

Il est possible de fusionner *"naming games"* et *"discrimination games"*. Comme pour les jeux lexicaux précédents, les agents doivent parvenir à s'accorder sur un vocabulaire pour décrire un ensemble d'objets ou de situations du monde. Lors d'une interaction, le premier agent choisit un objet et le désigne au second agent de façon "non-linguistique". Les deux agents pratiquent

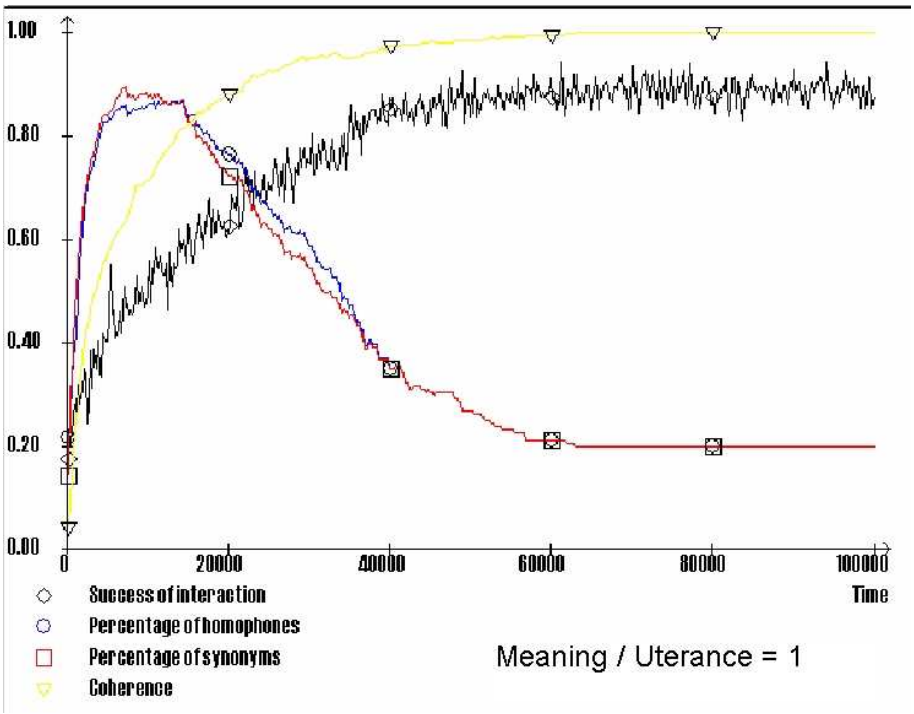

FIG. 3.8 – Impact du contexte sémantique et homophones : interactions à un mot et ratio concepts sur mots égal à 1

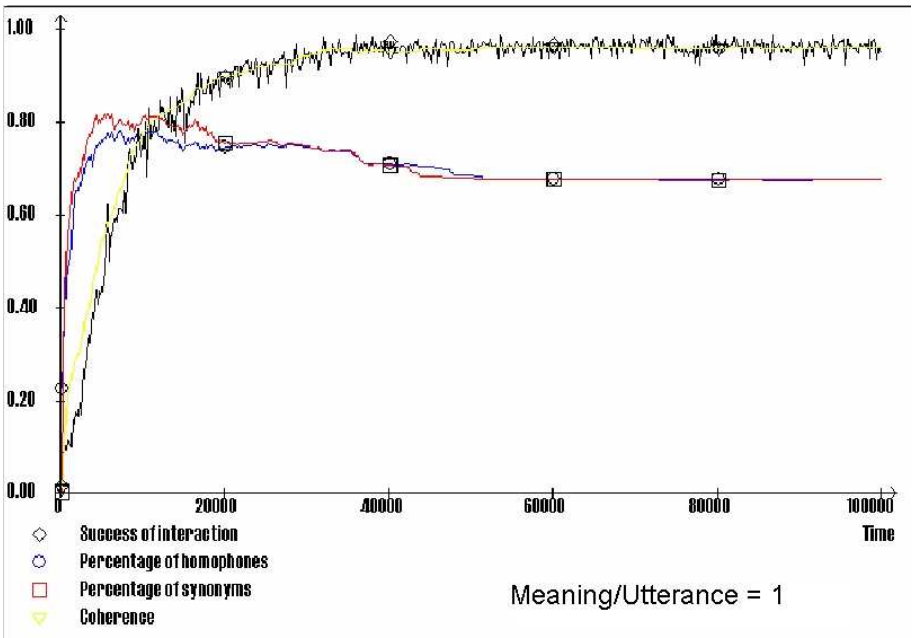

FIG. 3.9 – Impact du contexte sémantique et homophones : interactions à deux mots et ratio concepts sur mots égal à 1

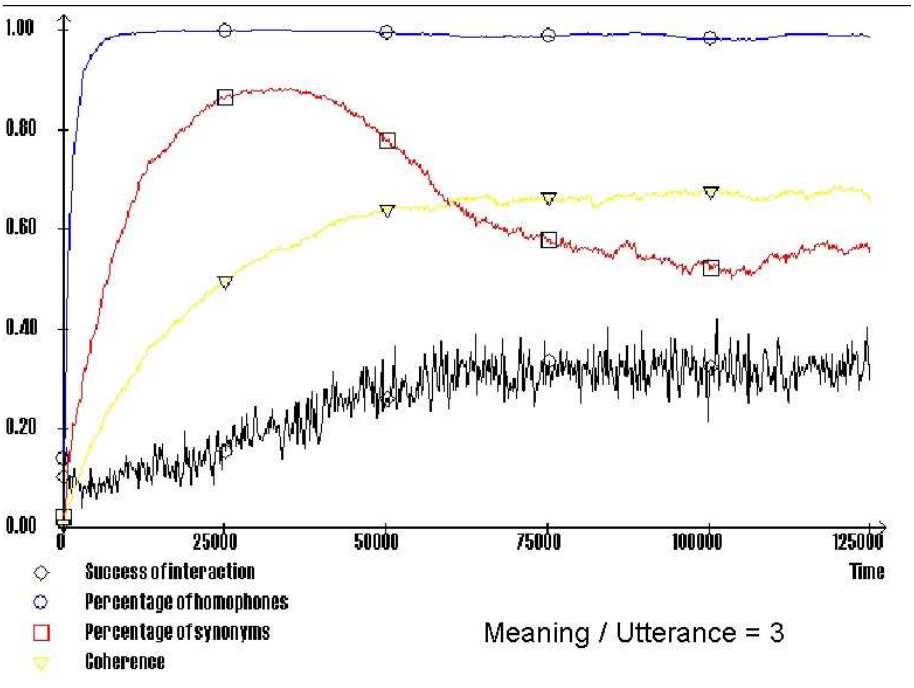

FIG. 3.10 – Impact du contexte sémantique et homophones : interactions à un mot et ratio concepts sur mots égal à 3

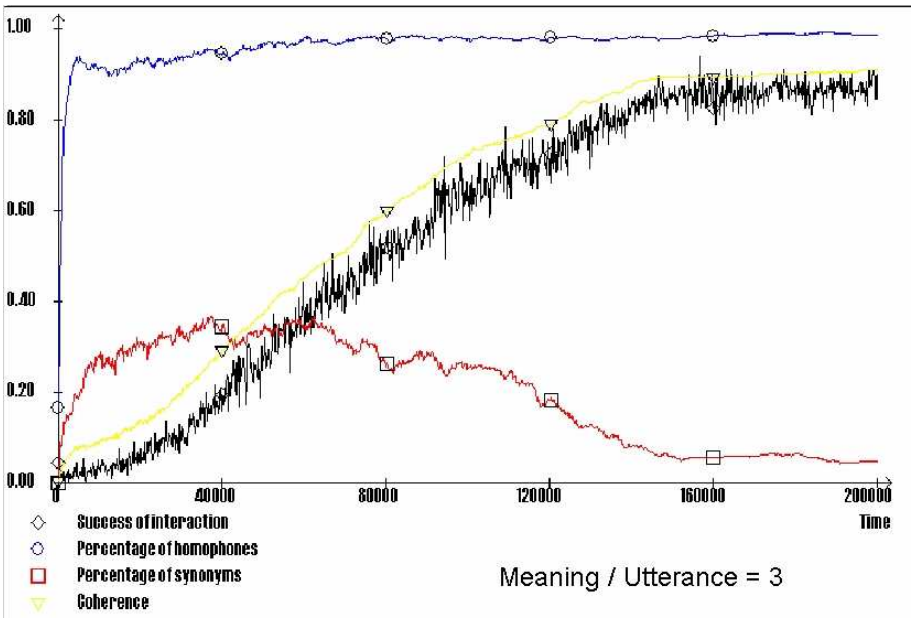

FIG. 3.11 – Impact du contexte sémantique et homophones : interactions à deux mots et ratio concepts sur mots égal à 3

chacun un jeu de discrimination pour l'objet désigné. Le premier agent encode alors les traits extraits à l'aide des mots de son lexique (des mots nouveaux peuvent être créés en cas de manque), et les communique à l'autre agent. Celui-ci examine les correspondances de ces mots dans son propre lexique, et compare les traits extraits de l'input linguistique avec ceux produits par ses capteurs. En cas d'échec, les agents tentent d'accorder leur lexique [Steels, 1997b] (p. 6-7). Après un certain nombre d'interactions, une discrimination partagée par l'ensemble des agents émerge et peut résister à des ajouts d'objets, à un renouvellement des agents...

Univers réel et expériences robotiques. L'environnement et les objets manipulés par les agents peuvent être abstraits, comme dans le cadre d'une simulation informatique totalement logicielle. Steels pense néanmoins qu'une incarnation des agents dans un monde physique crée des phénomènes intéressants qui sont passés sous silence dans les simulations logicielles. En effet, la perception de l'environnement tout comme la transmission d'information ne sont pas à l'abri des imperfections de tout système physique, et des erreurs peuvent se produire.

L'expérience *Talking Heads* menée au laboratoire Sony CSL à Paris et dans différentes villes du monde est l'un des points d'orgue des expériences robotiques récentes [Steels, 1999]. Des agents virtuels, qui voyagent sur le réseau Internet, peuvent se téléporter et s'incarner dans des couples de "robots" (en fait constitués de caméras mobiles sur leur pied) présents dans les différentes villes. Une interaction mixte entre *naming game* et *discrimination game* prend place entre les deux agents incarnés au même endroit. Elle est soumise aux conditions particulières de l'environnement (en particulier l'éclairage, les propriétés des différentes peintures utilisées pour dessiner les différentes figures...) Comme pour les simulations informatiques "abstraites", les agents finissent par établir une convention lexicale pour les différentes figures géométriques, et ce malgré les variations et les imperfections du monde réel.

Une des rares critiques que nous pouvons citer ici contre ces simulations est le passage sous silence des motivations des agents à communiquer. Un postulat implicite est en effet que les agents souhaitent échanger de l'information et nommer les objets qui les entourent, alors que certains chercheurs estiment qu'il s'agit du problème principal à prendre en compte.

Au niveau phonologique. L'émergence des conventions a également été étudiée au niveau des systèmes de sons, avec bien sûr une abstraction des interactions différente de celle du cas lexical. Les premières expériences, menées par Bart de Boer, reposent sur l'algorithme suivant dénommé *"imitation game"* [De Boer, 1999] (p. 36) :

- choix de deux agents au hasard dans la population, un initiateur et un imitateur ;
- choix d'une voyelle de son inventaire par l'initiateur et prononciation de cette voyelle grâce à un modèle de tractus vocal ; ajout de bruit ;
- l'imitateur analyse le signal et le compare aux voyelles de son inventaire. Il choisit le son le plus proche (avec une distance en barks[27]) et synthétise le son correspondant qu'il émet en direction de l'initiateur ; ajout de bruit ;
- l'initiateur écoute le son reçu et compare la voyelle extraite à la voyelle émise : si les deux sont identiques, l'interaction est réussie ; dans le cas contraire, l'initiateur communique de façon extra-linguistique à l'imitateur l'échec de la communication.

[27]Les barks forment une échelle de valeurs qui reproduisent les perceptions vocaliques de l'oreille humaine.

Afin de parvenir à développer des systèmes de sons similaires chez les agents, les mécanismes suivants sont utilisés (p. 36-37) :

- pour chaque voyelle, l'agent retient le nombre d'utilisations et de succès lors des interactions. Le ratio de la première valeur sur la seconde fournit le taux de succès de la voyelle ;
- si une interaction est réussie, l'imitateur déplace la voyelle de son inventaire qu'il a choisie vers le signal reçue par l'initiateur, afin d'augmenter la cohérence de la population. Pour ce faire, et puisque un agent ne manipule que les paramètres de son modèle articulatoire, l'imitateur recourt à un processus de **babillage** (*"babbling"*) qui permet d'explorer de façon partielle l'espace articulatoire autour de la voyelle de l'inventaire. Cette dernière est alors remplacée par l'élément de l'investigation par babillage le plus proche du son reçu de la part de l'initiateur. Un paramètre ε permet de définir la taille de l'espace d'exploration du babillage ;
- en cas d'échec de l'interaction, si le taux de succès de la voyelle de l'inventaire est élevé, une nouvelle voyelle est créée au centre de l'espace acoustique, avec pour objectif futur de la rapprocher de la voyelle émise par l'initiateur. Si le taux de succès est faible, la voyelle choisie par l'imitateur sera rapprochée de la voyelle de l'initiateur comme précédemment. Un paramètre θ_s permet de définir la catégorisation des taux de succès.

Les systèmes obtenus grâce à ces algorithmes conduisent à l'émergence de systèmes vocaliques qui sont assez proches des systèmes observés dans les langues du monde, en particulier en ce qui concerne les aspects de contrastes.

Suite aux travaux de Bart De Boer, certains chercheurs ont infléchi le problème en s'attaquant à une instance plus difficile de l'émergence des conventions sonores : l'émergence de systèmes syllabiques, et non plus seulement vocaliques. Les travaux de Pierre-Yves Oudeyer dans ce domaine sont déjà très aboutis, et suivent principalement deux directions. La première consiste à modéliser les phénomènes de co-articulation entre phonèmes à l'aide d'un modèle d'inertie des articulateurs. En comparant une situation où les agents mémorisent des sons complexes de façon holistique à une situation où ils mémorisent au contraire les composants de ces sons, les avantages de la seconde approche sont mis en valeur. Les résultats de Bart de Boer sont reproduits, mais cette fois dans un complexe de sons beaucoup plus riche [Steels and Oudeyer, 2000]. A l'aide de modèles sophistiqués de la cochlée et du tractus vocal (modèle de Cook), Oudeyer a également montré que le modèle dynamique précédent conduit à des systèmes syllabiques proches de ceux rencontrés dans les langues du monde [Oudeyer, 2001b].

Une seconde direction est le recours à des cartes neurales pour rendre compte de l'émergence d'un codage phonémique. Oudeyer pense qu'il est possible de rendre compte de ce codage à l'aide des simples propriétés dynamiques et de classification de cartes biologiquement plausibles, et ce sans pression de sélection [Oudeyer, 2001a].

Les deux approches précédentes sont enfin fusionnées dans les modèles les plus récents, où des cartes neurales dynamiques permettent l'encodage et la mémorisation des trajectoires articulatoires et acoustiques mentionnées plus haut [Oudeyer, 2002].

Pour conclure, citons les travaux d'Emmanuelle Perrone, qui a considéré les liens entre niveaux lexical et phonologique, par le biais de contraintes lexicales (en terme de nombres de mots) sur les systèmes phonologiques [Perrone, 1999].

Au niveau syntaxique. L'émergence de conventions de caractère syntaxique est l'un des domaines les plus récents abordés par les modélisateurs.

John Batali a approché ce problème grâce au paradigme des réseaux de neurones artificiels, et plus particulièrement en recourant à des perceptrons multi-couches récurrents, qui permettent de traiter et mémoriser des motifs pourvus d'une dimension temporelle [Batali, 1997]. A l'aide de simulations multi-agents où chaque agent est en fait un réseau de neurones, l'auteur étudie la communication de signaux (composés de plusieurs lettres parmi a, b, c ou d) permettant l'expression de structures de l'espace sémantique (ces structures sont sous forme de prédicats à deux arguments ; le prédicat comme les arguments sont représentés par des structures de quelques bits : 000, 010, 011...). L'entrée du réseau correspond aux signaux de communication, et la sortie aux composants sémantiques. Le réseau ne pouvant fonctionner que dans un seul sens (de l'entrée vers la sortie), un mécanisme de tests des différentes lettres pour former le signal est utilisé pour déterminer progressivement la séquence du signal (p. 411). Le décodage du signal ne pose par contre pas de problème. Lors d'une interaction, l'agent récepteur a connaissance du prédicat exprimé par l'agent émetteur, et son réseau peut être entraîné à partir du signal de communication et du prédicat. L'auteur montre comment dès lors une structuration *a priori* de l'univers sémantique (sous la forme de prédicats) émerge progressivement de façon réfléchie dans la structure des signaux de communication.

Plus récemment, Batali a eu recours à un modèle symbolique (de haut niveau) pour étudier l'émergence de conventions syntaxiques entre des agents. L'idée est ici que les signaux compositionnels sont produits par l'émetteur en considérant *a priori* la façon dont le récepteur peut les interpréter. Réciproquement, l'émetteur analyse les signaux compositionnels en considérant comment ils ont pu être encodés par l'émetteur [Batali, 2000]. En connaissant à la fois le signal et le sens du message émis, le récepteur tente ainsi de découvrir les conventions linguistiques utilisées par l'émetteur. Une convention syntaxique ("négociée") apparaît après un certain nombre d'interactions, et différentes régularités ou contraintes syntaxiques émergent ; elles ressemblent en partie aux structures observées dans les langues du monde.

Steels a critiqué ces modèles en arguant du fait que les individus n'ont pas accès dans la réalité aux contenus de l'esprit des personnes avec lesquelles ils communiquent (*"no mind-reading"*). Il a lui même tenté de corriger ce biais à l'aide d'un modèle plus complexe [Steels, 2000]. Toujours dans le cadre robotique souvent employé par Steels, ce modèle repose sur une architecture en deux couches. Une première couche est constituée de réseaux de neurones qui réalisent le passage des données extérieures à des données symboliques : certains réseaux servent à comparer des éléments à un prototype, d'autres à comparer une des dimensions des éléments par rapport à une valeur moyenne... Ces dernières données vont ensuite être manipulées par les algorithmes de la seconde couche, qui vont encoder linguistiquement les primitives perceptives. Différentes stratégies (spécifiées par avance) peuvent être envisagées pour cet encodage : une lexicalisation totale des différentes primitives (un mot pour chaque objet, un mot par relation entre objets...), l'utilisation de l'ordre des mots de la phrase, une méthode mixte ... Comme les conceptions sémantiques des agents ne sont pas partagées, l'agent récepteur d'un message produit différentes inférences sur le contenu sémantique de celui-ci. Chaque association entre un mot et une primitive possède un score comme pour les *naming games* évoqués plus hauts. De nombreuses hypothèses émergent par le biais des inférences du récepteur, qui tente de privilégier celles qui sont consistantes de façon interne, en accord avec les scores des associations lexicales, et enfin en accord avec les données qu'il peut acquérir sur le monde qui l'environne. Sur la base de l'association la plus probable, le récepteur modifie alors son lexique et sa grammaire. Des

routines existent qui permettent d'apprendre ou de créer des nouveaux mots. Des mots peuvent être utilisés non seulement pour les primitives, mais également pour représenter des structures composées de primitives.

Les premières simulations effectuées avec les stratégies de lexicalisation intégrale, d'utilisation de l'ordre des mots ou encore d'économie dans les expressions, ont conduit à l'émergence d'une convention entre les agents.

Pour finir, Simon Kirby recourt à des méthodes symboliques pour l'étude de l'émergence des structures syntaxiques dans une population d'agents. Le but de l'étude est l'observation de l'émergence de la compositionnalité à partir de signaux holistiques pour l'expression de prédicats [Kirby, 2000]. Le présupposé est ici que les agents tentent de décomposer les signaux holistiques. Les agents entament les simulations sans posséder de mots, mais peuvent créer ceux-ci pour refléter les structures syntaxiques :

"Briefly, the invention algorithm used by the simulation generates strings where the speaker has no grammatical structure, but for meanings that can be partially expressed with a particular grammar will only randomise those parts of the string that are known by the speaker *not to correspond to expressible meaning."* [Kirby, 2000] (p. 6)

Le mécanisme d'apprentissage (dit mécanisme d'induction) est le suivant : à la réception d'un couple sens-forme linguistique, le récepteur ajoute dans sa grammaire la correspondance holistique entre la forme et le sens. Une deuxième étape consiste à inspecter la grammaire de l'agent pour voir s'il est possible de fusionner la règle introduite précédemment avec d'autres déjà présentes en mémoire. Ceci permet de compresser l'information en mémoire.

Les agents sont placés sur un anneau et possèdent donc chacun deux voisins avec lesquels ils peuvent communiquer. Lors de l'évolution du système, les agents peuvent être remplacés de façon aléatoire, et la probabilité qu'un agent traite l'ensemble des prédicats possibles est extrêmement faible.

L'observation du système permet de voir qu'après une certaine période où le nombre de prédicats exprimables en moyenne par un agent reste assez faible, ce nombre augmente de façon significative pour atteindre pratiquement la totalité des prédicats de l'espace sémantique. Ceci traduit en fait l'apparition d'agents possédant des grammaires compositionnelles, qui permettent de former l'ensemble des prédicats possibles, même si ceux-ci ne sont pas entendus par l'agent au cours de sa vie.

Modélisation des processus d'acquisition ; interaction entre phylogenèse et ontogenèse

Plusieurs travaux de modélisations portent sur l'acquisition par un système d'un certain nombre de structures syntaxiques. Cette acquisition est parfois examinée en lien avec la phylogenèse, afin de faire ressortir comment une telle étape a pu se mettre en place sous une certaine pression de sélection.

Les travaux de John Batali ont ainsi mis en évidence que la *période critique* pour l'acquisition du langage (et d'ailleurs aussi pour d'autres comportements, que ce soit chez l'homme ou l'animal) peut être expliquée non pas par un processus de maturation exogène, mais par une dégradation des performances d'un réseau neuronal aux poids initiaux spécifiés de façon innée mais soumis à des entrées en désaccord avec sa configuration initiale [Batali, 1994].

A l'aide de réseaux de neurones récurrents devant apprendre à reconnaître les chaînes de caractères générées par un sous-ensemble des grammaires libres de contexte (*context-free grammars*), Batali a simulé une évolution darwinienne en sélectionnant les réseaux les plus performants au cours de l'apprentissage. Le génotype des systèmes, transmis et soumis à des mutations, était constitué des poids initiaux des réseaux de neurones (on retrouve en fait ici une forme dérivée de l'apprentissage pour les réseaux par algorithmes génétiques). Après un certain nombre de générations, les réseaux parviennent très bien à apprendre les régularités des grammaires considérées. Les poids initiaux ont évolué pour correspondre partiellement aux grammaires en jeu. Batali constate alors qu'un entraînement sur des données non structurées puis sur des données structurées par une grammaire en jeu, ou un apprentissage portant d'abord sur une grammaire particulière puis sur une autre conduit à une dégradation des performances. Le réseau perd alors sa capacité "innée" à apprendre correctement les chaînes d'une grammaire. Ainsi, après une période sans données structurées au cours de laquelle le réseau est entraîné sur des entrées sans relation avec le langage cible, la capacité d'acquisition du réseau se dégrade, et celui-ci devient incapable d'acquérir le type de langage pour lequel il était préparé initialement. Ceci peut expliquer l'existence d'une période critique.

A côté de ces travaux échafaudés sur des bases sub-symboliques et ne postulant aucune contrainte formelle innée relative au langage à acquérir, ainsi que parallèlement aux travaux reposant sur une sélection culturelle ou une absence de sélection des structures typologiques, les partisans de la théorie de la Grammaire Universelle tentent également de simuler les processus à l'œuvre dans leurs modèles. L'acquisition est ici basée sur un ensemble de paramètres à ajuster, grâce en partie à des modèles probabilistes qui permettraient à l'enfant de déterminer quelles hypothèses de la Grammaire Universelle sont celles employées dans la langue qui l'environne. Ted Briscoe propose un exemple de ce type d'apprentissage, basé sur un modèle probabiliste de type bayésien (modèle basé sur des probabilités conditionnelles) [Briscoe, 2002].

Toujours dans le cadre des grammaires universelles, Charles Yang refuse les modèles d'apprentissage statistique généraux sans *a priori* sur l'espace de recherche des formes linguistiques, car ils sont d'une part computationnellement très coûteux, et d'autre part ne respectent pas le principe de compatibilité développementale que devrait exhiber tout modèle d'acquisition. Celui-ci devrait en effet montrer un apprentissage temporellement congruent avec celui des enfants [Yang, 1999]. Parallèlement à ce premier refus, l'auteur émet également des doutes sur les modèles transformationnels, qui ne permettent qu'une seule hypothèse grammaticale chez l'enfant à un instant donné, ce qui entraîne une incohérence vis à vis des irrégularités produites par les enfants et de la gradualité de leur apprentissage. Yang propose ainsi un modèle *variationnel* où plusieurs hypothèses grammaticales peuvent coexister et être en compétition pour l'acquisition de la langue cible.

Modèles de diversité et de changements linguistiques

Une partie des simulations, si elles s'intéressent à la façon dont les agents partagent une norme linguistique, axent plus leurs hypothèses et leurs résultats sur des phénomènes qui sont rencontrés dans les langues actuelles que sur des problèmes d'émergence. Ceci peut concerner tant le problème de la propagation d'un changement (son implémentation), que l'existence d'universaux.

Les modèles de Daniel Nettle que nous avons déjà amplement décrits rentrent dans ce cadre. Ils abordent en particulier le problème de l'implémentation des changements dans la commu-

nauté [Nettle, 1999c], en soulignant en particulier l'importance de la taille de la population [Nettle, 1999a] et la question de la diversité linguistique [Nettle, 1999b].

Simon Kirby s'est intéressé à l'émergence de certaines structures typologiques, ainsi qu'à celle de certains universaux implicationnels [Kirby, 1997]. En postulant *a priori* des contraintes cognitives entre ordre du verbe et de l'objet et position de la tête dans les phrases (au sens des théories de la Grammaire Universelle : phrase nominale, phrase verbale...) (voir chapitre 2), l'auteur montre comment des individus placés sur une grille et interagissant avec leurs proches voisins adoptent l'une des quatre possibilités de combinaison. Les simulations font apparaître différentes régions spatiales où les agents adoptent tous l'une des deux combinaisons les plus compatibles avec les contraintes précédentes. Aux frontières entre régions de combinaisons opposées, sous l'action de pressions contraires, des agents adoptent des systèmes moins en adéquation avec les contraintes. Kirby rend compte grâce à ces résultats de l'existence de tendances ou d'universaux dans les langues du monde.

Dans le cadre de la grammaire universelle, Niyogi et Berwick proposent enfin un modèle générique et dynamique du changement linguistique, dont les propriétés reposent sur l'algorithme d'apprentissage utilisé par les enfants pour acquérir la langue cible. A l'aide de distributions de grammaires dans une population et d'algorithmes d'apprentissage particuliers (hérités de théories de type innéiste), les auteurs appliquent leur modèle à des cas de la linguistique historique [Niyogi and Berwick, 1997].

Modélisations de la sélection naturelle pour l'émergence du langage

Un autre type de simulations que nous souhaitons brièvement évoquer ici concerne les modèles de théorie des jeux qui s'intéressent au problème de l'altruisme sous ces différentes formes (voir chapitre 1).

Nombre de ces simulations, comme celles de Jean-Louis Dessalles, s'appuient sur les algorithmes génétiques introduits plus haut, afin de tester si l'évolution darwinienne permet ou non l'émergence de certaines catégories d'individus honnêtes ou tricheurs. Différents coûts numériques sont définis pour les différentes actions que peuvent entreprendre les agents, et le succès reproductif d'un individu est fonction de ses performances : plus ses coûts sont faibles et ses gains importants, plus nombreux seront ses descendants.

Nowak, sans recourir aux algorithmes génétiques, tente de déterminer les valeurs critiques de paramètres permettant le passage d'un stade non syntaxique à un stade syntaxique dans la communication humaine. Ces modèles mathématiques ne permettent cependant pas d'étudier la dynamique des transitions. L'émergence d'un code compositionnel est ainsi étudiée par opposition à un codage par signaux holistiques, et un seuil sur le nombre d'éléments sémantiques complexes à exprimer est déterminé, au delà duquel la sélection favorise l'émergence [Nowak et al., 2000]. De la même façon, Nowak étudie également comment un système de signaux peut être associé à un ensemble d'éléments sémantiques de façon efficace [Nowak and Krakauer, 1999], ou encore les conditions d'existence pour l'émergence d'une Grammaire Universelle au cours de l'évolution [Nowak et al., 2001].

Emergence de la symbolisation

Parallèlement à l'étude de l'émergence ou de l'évolution de conventions linguistiques, certains chercheurs se penchent sur une étape antérieure, à savoir l'émergence de la symbolisation.

Le modèle proposé par Cangelosi et Parisi nous semble l'un des plus intéressants à ce sujet [Cangelosi and Parisi, 1998], puisqu'il semble répondre à la critique mentionnée plus haut sur la motivation des agents. Les auteurs placent un ensemble d'agents dotés de capacités abstraites visuelles et locomotrices dans un univers où ils doivent subsister. Pour ce faire, ils doivent consommer des champignons qui se trouvent répartis dans l'environnement. Une partie de ces champignons est comestible et rapporte des points à ceux qui les consomment, quand d'autres sont au contraire empoisonnés et font perdre des points aux agents qui s'en nourrissent. Le modèle allie réseaux de neurones (de type perceptron multi-couches) pour la gestion du comportement des agents et algorithmes génétiques qui, comme précédemment, servent à sélectionner les meilleurs individus selon leurs performances. Ce sont ainsi des populations de réseaux de neurones qui sont soumises aux lois de l'évolution.

Chaque réseau multi-couches accepte sur sa couche d'entrée un input visuel (composé de 10 unités, permettant de spécifier les différents champignons vénéneux ou comestibles) et un input pour des signaux de communication (3 unités). En sortie, cinq neurones permettent de coder les mouvements que doit effectuer l'agent (pour se rapprocher ou s'éloigner des champignons), et d'autres permettent d'émettre un signal (3 unités).

Chaque agent possède une durée de vie fixe. Il gagne 10 points pour chaque champignon comestible ingéré, et perd 11 points s'il consomme un champignon vénéneux. Une population initiale de 100 agents évolue par phases. A chaque épisode, les agents les plus performants sont sélectionnés pour produire la génération suivante, et du bruit est ajouté à une partie des connexions des réseaux (10%).

Différentes simulations sont menées. Une en particulier oblige les agents à s'appuyer sur les signaux de communication d'autres agents pour savoir si le champignon est ou non comestible et s'ils doivent s'approcher de lui pour le consommer ou non. Pour chaque situation, deux agents sont tirés sont hasard, et tous jouent donc un grand nombre de fois les "indicateurs" et les "indiqués". On observe alors l'émergence d'un système de communication honnête, qui permet aux agents qui reçoivent des messages de s'approcher correctement des champignons comestibles. Le langage utilisé permet ainsi d'améliorer les performances des agents. Les raisons de son émergence ne sont toutefois pas immédiates, puisqu'il n'est *a priori* pas nécessaire à l'émetteur des signaux de communication.

L'explication de l'émergence d'un tel système de communication s'explique peut-être par le fait que les agents jouent tour à tour le rôle d'émetteur et de récepteur, et que c'est la même structure qui est utilisée dans les deux cas, à savoir le perceptron multi-couches.

Notons ici que le modèle de Cangelosi et Parisi ne permet pas de conclure à l'émergence d'un altruisme réciproque. En effet, le nombre de champignons est suffisant pour tous les agents, et il n'existe ainsi pas de compétition entre eux. Il nous paraîtrait intéressant de reproduire le modèle et de l'étendre en examinant l'effet de la compétition sur le code de communication partagé (observerait-on des agents qui trompent délibérément leurs prochains pour bénéficier de plus de nourriture ?).

Parfois en collaboration avec Cangelosi, Steven Harnad s'intéresse à l'émergence de la symbolisation dans un contexte moins naturel que le précédent. A partir de modèles à base de réseaux

de neurones de type perceptron multi-couches, il développe l'idée du vol symbolique (*"symbolic theft"*), qui correspond à la possibilité d'acquérir de nouveaux concepts à partir de concepts pré-existants, sans qu'une expérience sensorielle directe de ces nouveaux éléments soit nécessaire [Cangelosi et al., 2002].

3.1.5 Discussion

Il nous paraît utile de dégager ici quelques points fondamentaux des débats sur la modélisation informatique de l'émergence du langage. Ils nous permettront en effet de mieux contextualiser nos propres modélisations par la suite.

Accès au sens

En ce qui concerne l'émergence de la syntaxe, un des points importants pour l'apparition de conventions entre agents concerne l'*accès au sens*. Lorsqu'un individu communique de l'information à un autre, il le fait en effet sur la base de représentations sémantiques internes qui ne sont pas accessibles au récepteur, ce que Steels résume souvent par la formule *"no mind-reading"*. Comme nous l'avons vu, certaines modélisations outrepassent parfois purement et simplement cette restriction, mais d'autres insistent sur la possibilité de détecter le ou les concepts manipulés par le locuteur par des moyens extra-linguistiques et le partage de l'environnement. Sans cette possibilité de partage au moins partiel et ambigu des représentations, il paraît difficile de pouvoir construire une convention quelconque.

Pointer un objet du doigt est une façon simple de désigner à un individu un élément particulier sur lequel on va ou l'on a communiqué. Toutefois, il n'est pas toujours évident d'identifier le concept précis : si l'on pointe par exemple un objet composite, comment être sûr que l'on pointe l'objet dans sa totalité ou seulement l'une de ses composantes ? Si le problème semble aisé à résoudre pour des adultes, il est délicat pour le très jeune enfant dont l'espace conceptuel tout comme l'espace linguistique est en construction.

Une façon réaliste de représenter l'accès au sens par le biais de l'environnement est de recourir à des agents incarnés (des robots), dont les senseurs nécessairement non idéaux conduiront à des ambiguïtés lors des interactions (par exemple pour l'identification de plages de couleurs par des capteurs de lumière). Une autre possibilité est de simuler l'imperfection de la transmission sémantique par un "bruit sémantique" incorporé dans le modèle (voir par exemple les notions de portée et de focus sémantiques (*"meaning scope"* et *"meaning focus"*) de Steels [Steels and Kaplan, 1998]).

Il arrive souvent que l'ajout de bruit dans le modèle ne perturbe que peu l'émergence des caractéristiques qui apparaissaient en absence de bruit. Ce dernier est ainsi souvent un indicateur de robustesse du processus d'émergence. Il est dès lors légitime de s'interroger sur l'influence d'un accès au sens imparfait dans les processus d'émergence des conventions. Deux arguments peuvent être invoqués pour légitimer cet aspect : tout d'abord, l'utilisation d'un feedback extra-linguistique plutôt qu'un accès direct au contenu de l'"esprit" du locuteur peut entraîner une modification de la dynamique d'interaction qui, si elle paraît superficielle, peut avoir des conséquences importantes de par la non-linéarité des phénomènes en jeu. Ensuite, la présence de bruit peut venir jouer non pas sur les aspects qualitatifs des processus d'émergence, mais sur des aspects plus quantitatifs : temps de convergence, existence d'ambiguïtés ou de synonymes [Steels and Kaplan, 1998]...

Concluons par l'idée qu'un accès imparfait au sens aura d'autant plus de conséquences que les structures linguistiques manipulées seront abstraites : simples associations entre sens et formes phonétiques, ou au contraire structures d'organisation de la phrase (ordre des mots...) plus difficiles à atteindre *via* un feedback extra-linguistique.

Contraintes et structures

Différentes contraintes physiologiques ou cognitives sont le plus souvent prises en compte dans les modèles que nous avons présentés : contraintes en perception et en production à l'aide de modèles du tractus vocal ou de l'oreille, contraintes mnésiques grâce à des modèles de mémoire limitée... Toutefois, les contraintes structurelles du langage sont rarement prises en compte, si ce n'est dans des cas simples comme la compétition entre plusieurs mots d'un lexique ou plusieurs règles grammaticales pour l'expression d'un sens. Encore une fois, ceci est un constat et non pas un reproche, puisque les simulations ne se préoccupent généralement pas de différentes structures typologiques en même temps. Kirby a toutefois présenté plusieurs modèles mettant en jeu des contraintes entre structures afin d'observer l'émergence d'universaux ou de tendances implicationnels [Kirby, 1997].

Un autre phénomène générateur de contraintes est parfois absent des simulations, surtout dans les situations d'émergence où il n'est pas toujours jugé très pertinent : il s'agit de l'hétérogénéité de la population. Dans de nombreuses simulations dues à Steels, Kirby ou Batali, la population est le plus souvent homogène, et les interactions ont lieu de façon identique entre tous les individus. Ceci conduit en particulier au niveau du lexique à la compétition entre mots pour un même sens, et la sélection d'un seul d'entre eux après un temps plus ou moins long. Une hétérogénéité dans la population peut conduire à la préservation de plusieurs formes différentes, et à des phénomènes dynamiques intéressants. Elle peut-être envisagée de différentes manières : par le biais d'une distribution spatiale des individus comme dans [Kirby, 1997], par une distribution sociale comme dans [Nettle, 1999c], par la simulation de populations en contact comme dans [Marsico et al., 2000]... On constate par exemple qu'avec une distribution spatiale, différentes stratégies linguistiques peuvent co-exister dans une population.

Suivant la structure adoptée pour l'hétérogénéité de la population, les conséquences sur la dynamique d'évolution du système seront différentes. Le modèle le plus courant est celui d'une distance euclidienne entre les agents, qui sera proportionnelle à la force de leur interaction. C'est cette distance qui est adoptée le plus souvent dans les modèles spatiaux (où l'utilisation d'un tore plutôt qu'une grille fermée pour disposer les agents permet d'éviter les effets de bord et donc les dissymétries entre agents), et aussi dans les modèles sociaux de Nettle. Toutefois, de très nombreuses distances peuvent être définies à l'aide de métriques différentes. Différentes hétérogénéités peuvent également être prises en compte simultanément, comme par exemple une distribution sociale et spatiale. Notons ici que ce sujet de l'hétérogénéité d'un ensemble d'éléments fait écho au réductionnisme et à l'auto-organisation tels que nous les avions introduits au chapitre 2.

D'une façon générale, les phénomènes sociaux sont peu pris en compte dans les simulations actuelles sur l'émergence et l'évolution du langage. Comme nous l'avons vu au chapitre 2, il semble cependant que ceux-ci jouent un rôle fondamental dans l'évolution du langage, et également dans son émergence. Une réflexion sur ce point est nécessaire, avec une définition plus précise du type d'interaction entre les agents (avec par exemple des liens forts ou faibles comme chez Milroy). Il semble ici qu'une métrique euclidienne ne soit pas la plus adaptée pour représen-

ter les situations réelles. Comme nous l'avons déjà souligné, nous pensons que c'est un mauvais choix de distance sociale qui explique les résultats de Nettle et nécessite l'existence d'individus hyper-influents pour observer un changement au niveau populationnel. Nous reviendrons sur ce point au cours du chapitre 6.

Emergence et évolution

Il est important de ne pas confondre modèle d'émergence et modèle d'évolution, même si ceux-ci semblent parfois reposer sur les mêmes mécanismes. La distinction entre les deux est d'ailleurs à rapprocher de l'ensemble des remarques précédentes. Dans le premier cas, il s'agit d'observer l'émergence d'une convention, et les facteurs primordiaux une fois la convention établie sont alors moins pertinents.

Dans les modèles d'émergence, les agents débutent généralement la simulation avec un répertoire de constructions linguistiques vide. Celui-ci se remplit progressivement au cours des interactions des agents entre eux. Un flux d'agents peut être mis en œuvre, et de nouveaux individus arriver dans un contexte linguistique déjà formé. Néanmoins, ces agents sont identiques à ceux qui ont permis l'établissement de la convention, et ils peuvent créer de nouvelles structures de façon similaire et aussi importante.

Dans un contexte d'évolution, le principe est généralement de partir d'un contexte linguistique déjà formé, et d'envisager les évolutions du système à partir d'un état initial. Un point important à modéliser pour obtenir un modèle réaliste est l'arrivée de nouveaux agents, qui apprennent le langage mais avec une capacité d'innovation plus limitée que celle présente dans les modèles d'émergence. En effet, si les enfants et les adultes peuvent parfois innover, leur apprentissage de la langue cible est généralement assez fidèle et leur but n'est pas autant de faire émerger la convention que de l'acquérir et de la prolonger tout en y intégrant une partie de leur identité sociale. La distinction d'une phase d'acquisition et d'une phase adulte des agents est généralement utile à cette fin, et est souvent absente des modèles d'émergence. L'hétérogénéité de la population déjà évoquée est aussi un facteur important à prendre en compte, et la distinction entre enfants et adultes n'est finalement qu'une de ses composantes.

Mécanismes explicatifs et reproduction de la réalité

Le but d'un modèle est généralement de reproduire des phénomènes réels et par un processus de simplification et d'abstraction, d'en extraire les composantes pertinentes, c'est à dire qui permettent d'expliquer les caractéristiques de ces phénomènes.

Néanmoins, pour un phénomène dans la réalité, il peut parfois exister plusieurs mécanismes permettant de décrire en partie les caractéristiques observées. Dès lors, il faut être prudent lors de la réalisation d'un modèle, afin de ne pas proposer des mécanismes explicatifs qui ne reflètent pas la réalité. Bien sûr, l'erreur est toujours possible, mais certains cas semblent parfois être invalides. Si nous reprenons l'exemple de Steels sur les phénomènes stochastiques qui pèsent sur l'évolution du lexique [Steels and Kaplan, 1998], les synonymes apparaissent et peuvent demeurer stables uniquement en cas de stochasticité sur la forme des mots. La stochasticité peut donc être invoquée ici comme mécanisme explicatif de la stabilité des synonymes, mais il nous paraît plus pertinent de recourir aux notions de registres de discours et de contextes de communication pour expliquer comment différents synonymes peuvent persister dans la population. Ceci est d'ailleurs renforcé par le fait que le modèle de Steels ne rend pas compte de la stabilité des synonymes en cas

de stochasticité et de flux d'agents dans le système. Cette situation, qui devrait être proche de la situation réelle, est qualitativement différente puisque des synonymes parfois nombreux subsistent pour de nombreux concepts.

Conclusions préliminaires

Les modèles et les simulations informatiques ont permis de faire de grands progrès dans la compréhension de l'émergence des conventions linguistiques, à travers des phénomènes comme l'**auto-organisation** ou le caractère stochastique des interactions linguistiques.

Le paradigme est ainsi en plein développement, et l'on peut raisonnablement estimer que des modèles plus sophistiqués apporteront par exemple de nouveaux éléments de réponse sur l'émergence de la syntaxe, en particulier en lien avec des mécanismes cognitifs plus généraux.

Si le problème de l'origine est au cœur des débats, celui de l'évolution des langues est resté jusqu'à présent assez à l'écart des travaux de simulation. Daniel Nettle est l'un des rares à s'être penché spécifiquement sur la question *en tentant d'inclure des données de sociolinguistique*. Sa démarche nous semble importante et pertinente, car s'il est possible de simuler certaines évolutions linguistiques, un ancrage dans les conditions réelles d'utilisation du langage est indispensable pour reproduire et comprendre les mécanismes réels d'évolution. Cette démarche est bien sûr différente et complémentaire de celles visant à établir des conditions *nécessaires* ou *suffisantes* pour l'émergence de conventions ou l'adaptation de systèmes lexicaux ou phonologiques.

Parallèlement à la dimension cognitive du langage (qui s'exprime au travers de phénomènes comme la diffusion lexicale), c'est sur la dimension **sociale** du langage qu'il nous paraît important d'insister. L'utilisation de ce dernier comme outil de positionnement social semble en effet fondamental dans l'évolution des langues, et l'évolution des structures sociales des populations humaines au cours de l'histoire a très probablement joué sur la mise en place des familles de langues telles que nous les observons aujourd'hui. De telles structures sont rarement considérées dans les simulations d'émergence puisqu'elles ne semblent pas être alors au cœur du sujet, mais il semble important de les prendre en considération pour les étapes post-émergence, au vu des nombreux travaux de sociolinguistique qui les placent au centre du débat sur les changements linguistiques. Ceci passe en particulier par une prise en compte des schémas dynamiques liés à une structuration particulière de la population, par exemple en réseaux sociaux, et qui conduisent à des évolutions spécifiques des systèmes linguistiques.

Comme nous l'avons au chapitre 2, les caractéristiques structurelles du langage sont également fondamentales pour les changements linguistiques, et bien que des caractéristiques systémiques soient apparentes dans les simulations précédentes (interactions et auto-organisation des voyelles chez de Boer, interactions entre les mots du lexique pour un même concept...), elles s'inscrivent le plus souvent dans un contexte différent de celui introduit par Saussure et repris par les structuralistes du XXème siècle.

Les modèles que nous développerons au chapitre 6 tenteront d'aborder ces deux points.

3.2 Un outil de modélisation : la plate-forme LEMMingS

Any rumors that the world is coming to an end just because
I'm about to release a 1.0-version are greatly exaggerated.
Linus Torvalds (principal concepteur du système d'exploitation Linux).

Afin de réaliser nos simulations, un de nos besoins était un outil performant, tant en termes de puissance de calcul (et donc de rapidité) qu'en termes de traitement et de représentations des données. Plutôt que de faire un choix parmi les offres logicielles existantes, nous avons préférer développer nos propres programmes. Les raisons de ce choix sont d'abord présentées, avant une description plus précise de notre plate-forme.

3.2.1 Langages et outils logiciels

Lors de la description de l'approche modélisatrice dans la section précédente, nous avons détaillé différentes techniques et plusieurs paradigmes pour la modélisation des évolutions linguistiques et de l'origine du langage. Nous avons insisté sur le degré de mathématisation plus ou moins marqué des modèles et incidemment sur le type de simulations réalisées pour "incarner" un modèle.

Des langages dédiés aux mathématiques

Des logiciels (tels R, Mapple ou Mathematica, pour ne citer qu'eux) proposent des répertoires très étendus de fonctions mathématiques et de représentations graphiques, et se prêtent de façon naturelle à la réalisation de simulations reposant sur des outils mathématiques tels que les équations différentielles (tout en permettant la mise en œuvre d'autres techniques, parfois fort facilement). Les graphiques très élaborés et faciles à produire (représentations tridimensionnelles, utilisation judicieuse des plages de couleurs...) sont un avantage, bien qu'ils ne soient pas toujours nécessaires pour les paradigmes que nous avons introduits. Le défaut de tels logiciels est qu'ils sont généralement lents (parfois très lents), et que leur gestion de la mémoire ne permet pas toujours de manipuler de très grands volumes de données efficacement.

Quelques caractéristiques des langages de programmation

Parallèlement à ces langages de programmation de haut niveau dédiés aux mathématiques existent de nombreux langages moins spécialisés et de plus bas niveau, tels Java, Lisp, C, C++, Haskell, Visual Basic... Nous pouvons distinguer différentes caractéristiques importantes, qui nous ont guidé pour notre choix du langage de programmation de notre logiciel :

- le caractère compilé ou interprété du langage. Alors que la compilation d'un programme produit un code directement exécutable par la machine (code machine), l'interprétation nécessite une "machine virtuelle" qui va convertir à la volée un premier code en un second compréhensible par la machine. Ceci a pour effet de diminuer les performances par rapport aux langages compilés, mais présente un intérêt pour le partage du code intermédiaire entre plusieurs systèmes d'exploitation (comme les applets ou les fichiers d'extension ".class" du langage Java) ;

- le caractère fonctionnel ou non du langage. Les langages fonctionnels (Lisp, Caml, Haskell...) adoptent une sémantique particulière de programmation qui fait reposer toutes les opérations sur la réalisation de fonctions munies d'arguments. Cette approche, accompagnée de contraintes de typage des objets manipulés, permet de réaliser des programmes dont on peut contrôler de meilleure façon la robustesse et la correction (les langages "sûrs" peuvent être indispensable dans les environnements où un programme ne doit pas "planter" : centrale nucléaire, engins spatiaux...) ;
- l'orientation objet éventuelle du langage. Ce paradigme de programmation propose une structuration des programmes en termes d'objet. Une définition très générale des objets peut être celle d'entités conceptuellement et fonctionnellement closes en interaction. L'objet représente ainsi un concept (au sens large), en mettant l'accent sur son indépendance vis à vis des autres concepts. Pour ce faire, il est doté d'un certain nombre de composantes (visibles ou non par les autres objets), qui traduisent les propriétés du concept qu'il représente, et de fonctions qui permettent d'agir sur ces propriétés ou d'y accéder.

  De la même façon qu'il est possible de combiner les concepts ou de les faire dériver les uns des autres (le concept d'autruche dérive par exemple du concept d'oiseau en héritant de ces propriétés, et en possédant d'autres caractéristiques qui le rendent plus spécifique), un objet peut être constitué d'autres objets et hériter d'objets plus génériques que lui. Cet aspect se traduit de façon utile dans les langages de programmation orientés-objet, en permettant des économies dans l'écriture du code.

Il est aisé de se rendre compte de l'isomorphisme entre les notions d'agent et d'objet (l'agent étant une transposition de la notion d'objet au niveau de l'exécution du programme), ce qui rend les langages orientés-objet bien adaptés au développement de systèmes multi-agents.

Le "niveau" du langage est assez fortement corrélé au degré de performances obtenu et à la complexité de la tâche de programmation. Plus un langage est bas niveau, plus la charge de travail repose sur le programmeur et plus celui-ci doit faire attention à la correction de ces programmes ; les vérifications sont en effet moins nombreuses ou même absentes. La gestion de la mémoire est un bon exemple, puisque certains langages la prennent entièrement en charge, quand d'autres la laissent entièrement à la charge du programmeur (nombreuses sont les erreurs en C ou C++ imputables à l'oubli de déclarer un espace réservé en mémoire). En contrepartie de ces difficultés, les performances sont souvent bien supérieures, avec par expérience des gains d'un facteur pouvant aller jusqu'à 5 ou 10.

Présentation et choix du C++

Le langage C++ est l'évolution orientée-objet du langage C, et tous deux sont probablement les langages les plus utilisés depuis de nombreuses années. C'est un langage de bas niveau comparé à de nombreux autres (donc plus "difficile" à programmer), dépourvu initialement de possibilités de représentations graphiques, mais très performant. Son modèle objet n'est en fait que virtuel (les classes sont plus des conventions syntaxiques du langage, et sont transformées à un niveau rudimentaire de la compilation), ce qui participe à ses performances. Son succès ainsi que celui du C ont conduit à une optimisation des techniques de compilation, ainsi qu'au développement par les utilisateurs d'un très grand nombre de bibliothèques qui viennent combler l'aspect rudimentaire du langage de base (si on le compare à d'autres). Le *World Wide Web* permet un accès facile à ses bibliothèques, ainsi qu'à une documentation volumineuse.

Ces différentes caractéristiques et les besoins que nous avions pu concevoir pour nos simulations nous ont conduit à faire le choix final du langage C++ pour le développement de notre logiciel[28] :

- ses performances très élevées permettaient d'envisager au mieux des simulations avec un grand nombre d'agents ou des bases de données importantes ;
- les structures objet, avec leurs propriétés de compositionnalité et d'héritabilité, permettaient la stratégie de conception que nous allons développer dans le chapitre suivant ;
- l'utilisation d'une bibliothèque graphique nommée **Gtk+2.0** nous offrait la possibilité d'une utilisation du logiciel sous Windows et Unix/Linux avec de bonnes performances [GTK, 2002] ;
- enfin, notre intérêt pour la programmation objet et les langages bas-niveau et performants nous poussait dans cette direction.

Il nous faut mentionner l'influence sur le développement du logiciel de l'accès à la plate-forme *Babel*, créée par Angus Mac Intyre au laboratoire Sony CSL de Paris [Mac Intyre, 1998]. Sans avoir eu accès au code du noyau (écrit en Lisp), ce logiciel fut cependant notre inspiration pour différents aspects.

3.2.2 Description du logiciel

Le nom LEMMingS est l'acronyme de l'expression *Language Evolution Modelling and Monitoring System*. Le but premier de ce logiciel est la réalisation de simulations informatiques pour le test de divers modèles liés à l'émergence et à l'évolution du langage. Il peut néanmoins également être utilisé pour aborder des questions et des problèmes issus d'autres disciplines.

Lors du développement du logiciel, un objectif fut la poursuite de principes généraux menant à un produit utile et performant pour différents aspects. Nous détaillons tout d'abord ces principes, avant d'aborder la structuration naturelle du code qui en a découlé.

Principes généraux de développement

Le développement de notre logiciel s'est articulé autour de plusieurs principes qui nous paraissaient souhaitables pour son utilisation par différents utilisateurs et pour son utilité à long terme :

- facilité d'utilisation à l'exécution ;
- facilité de développement pour le programmeur ;
- possibilité forte d'évolution et d'ajout de fonctionnalités ;
- qualité du code.

Afin de répondre à ce cahier des charges, nous avons adopté de façon correspondante les stratégies globales suivantes :

[28] Le langage Java que nous avions utilisé pour une première version de notre logiciel est un langage très proche du C++, mais son caractère interprété le rendait trop lent en exécution pour des simulations mêlant base de données topographiques et grand nombre d'agents. La solution de définir des bibliothèques en C++ incorporées aux sources en Java s'est révélée peu efficace en terme de qualité et de gestion du code.

- création d'une interface graphique, utilisation préférentielle de la souris ;
- adoption d'une structure en deux couches, composée d'un noyau et de modules externes ;
- utilisation du modèle objet, composition et héritage des classes tant au niveau du noyau que des modules pour permettre les extensions ;
- forte modularité du code.

L'adoption d'une structure en deux couches repose sur l'idée suivante : créer tout d'abord une partie centrale dans le logiciel, un noyau qui contiendra l'ensemble des fonctionnalités de base, indépendantes des spécificités et des buts poursuivis par une simulation particulière, puis un ensemble de modules proposant des fonctions que l'utilisateur pourra intégrer ou non selon les besoins d'une expérience spécifique. Ceci permet bien sûr de ne pas inclure des éléments inutiles, et d'obtenir une structuration plus claire et élégante du code. En outre, les développements ultérieurs se font de façon incrémentale : chaque nouvelle fonctionnalité programmée pour les besoins d'une application est présente en tant que module indépendant, et peut-être réemployée et/ou enrichie pour des simulations futures.

Parmi les fonctionnalités de base contenues dans le noyau se trouvent les représentations graphiques et textuelles de base (qui peuvent être éventuellement héritées et enrichies dans des modules supplémentaires), l'interface graphique générale, les possibilités de sauvegarde, les scripts...

Il est à noter qu'il n'est pas nécessaire de recourir au noyau pour pouvoir utiliser les différents modules. Ceux-ci peuvent être incorporés en tant que bibliothèques dans tout programme C++.

La composition et l'héritage des objets permet non seulement une bonne structuration du code, mais également une réutilisation de celui-ci qui permet de gagner un temps considérable dans le développement. Ainsi, si deux classes (les descriptions des objets dans le code) proposent la même fonctionnalité (par exemple la détection du pointeur de la souris dans la fenêtre graphique), il sera plus économique de définir une classe réalisant cette opération, et de la faire hériter par les deux premières que d'écrire deux fois le même code. En outre, un mécanisme appelé *surcharge* offre les deux options suivantes :

- spécifier (déclarer) sans les définir, ou déclarer par défaut, les propriétés qu'une sous-classe offrira ou devra offrir. Par exemple, une *super-classe* définissant d'une façon générale les interfaces graphiques pourra spécifier que chaque sous-classe (définissant une interface graphique particulière, par exemple pour tracer des courbes) *devra* gérer les clicks de la souris. Elle pourra également proposer une gestion par défaut de ces clicks ;
- définir des exceptions à un comportement général. Pour reprendre l'exemple précédent, une sous-classe pourra redéfinir la gestion de la souris, et plutôt que d'adopter le comportement générique proposé par la super-classe, opter pour son propre mode de traitement. De façon intuitive, pour 1000 éléments dont 3 se comportent différemment des autres, il est plus facile d'édicter une loi générale pour tous les éléments, et de préciser les exceptions (définition intensive), plutôt que de spécifier le comportement des 1000 éléments individuellement (définition extensive). Il en est de même pour l'écriture d'un programme.

Le fait de pouvoir définir les spécificités de chaque cas particulier à partir d'une déclaration plus générale permet ensuite de ne raisonner que sur le cas principal (en termes de prise en

compte dans le code). Là encore, il y a économie dans l'écriture du programme. L'utilisateur peut ainsi définir ses propres extensions des représentations graphiques et textuelles de base : il suffit qu'elles répondent correctement au canevas générique de ses classes pour être reconnues et traitées par le noyau.

Description du noyau

Le noyau constitue comme son nom l'indique le cœur du logiciel LEMMingS. Il regroupe les fonctionnalités de base du logiciel.

Description des composants graphiques de représentation des données. Afin encore une fois de permettre une bonne évolution du logiciel, nous avons tenté d'appliquer le principe de modularité au noyau lui-même. Les différentes interfaces graphiques ou textuelles permettant de représenter les données sont ainsi définies sous formes de librairies qui sont incorporées au noyau, et peuvent être utilisées indépendamment de ce dernier. L'utilisateur reçoit dans ce dernier cas la charge de la gestion des différentes fonctions : quand faut-il ajouter de nouvelles données, quand rafraîchir l'image présentée à l'écran...

Les représentations graphiques de base présentes dans le noyau sont actuellement les suivantes :

- un graphe permettant de dessiner des courbes en fonction du temps. Ces courbes sont en fait des ensembles de points (nous continuerons d'employer ce terme de façon erronée par la suite par commodité), qui peuvent ou non être connectés entre eux. Le mode par défaut est une compression horizontale de la courbe au fur et à mesure de la progression temporelle de la simulation, mais il est possible de représenter la situation sur l'intervalle de temps de son choix. Plusieurs courbes peuvent figurer sur le même graphe, à l'aide d'un jeu de couleurs et de figures pour les identifier. L'échelle des ordonnées peut être définie pour chacune d'entre elles. Une légende sous le graphe fournit une légende pour chaque courbe ;
- un graphe permettant de représenter des ensembles de points ou de vecteurs dans un espace bidimensionnel. Ce graphe permet d'afficher par exemple des populations d'agents évoluant dans un monde à deux dimensions, comme c'est très souvent le cas dans les simulations sur l'origine du langage lorsque l'on souhaite observer l'émergence de motifs spatiaux. Là encore, différentes populations peuvent être affichées avec différentes couleurs et figures. Il est également possible de "zoomer" à volonté sur une région de la carte. Un autre élément est la possibilité d'ajouter en arrière-plan une image représentant les valeurs prises par une variable continue dans l'espace bidimensionnel. Cette image se conforme bien sûr aux éventuels zooms, et permet par exemple de représenter une analyse spectrale ou encore une carte topographique terrestre (voir ci-dessous). Une légende résume ici aussi les significations des différents ensembles d'éléments affichés. Une grille peut-être sur-imposée en avant-plan pour une meilleure indication des échelles de valeurs, et la position de la souris ainsi que l'éventuelle valeur en un point sont également indiquées (s'il existe bien une variable continue qui prenne une valeur en chaque point) ;
- un affichage pour des lignes de texte. Ici encore, en jouant éventuellement sur les couleurs, il est possible d'afficher différents textes précédés d'une légende. L'affichage contient une fenêtre déroulante qui s'adapte automatiquement à la longueur du texte total à présenter ;

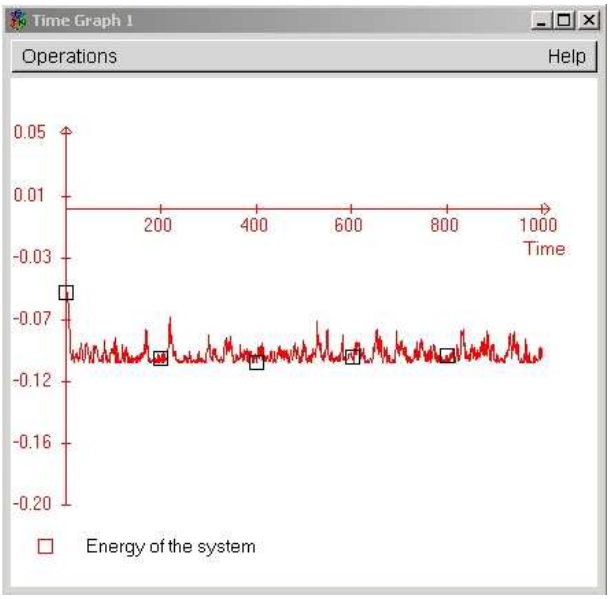

FIG. 3.12 – Le composant graphique **TimeGraph**, pour l'affichage de courbes de fonctions dépendantes du temps

- un module permettant la sauvegarde de lignes de texte vers des fichiers. Ce module est identique au précédent, mais la sortie texte se fait désormais dans un fichier.

Les figures 3.12, 3.13 et 3.14 illustrent les trois premiers composants précédents. Pour chacun d'entre eux, il est possible de sauvegarder la fenêtre graphique sous forme d'une image **png** (*Portable Network Graphics*), un format graphique répandu). Ceci permet ensuite d'insérer facilement les résultats graphiques dans des rapports ou articles, et la taille des fenêtres est modifiable pour s'adapter au mieux à la mise en page du document cible. La sortie de valeurs dans des fichiers texte rend possible des traitements ultérieurs et en particulier des analyses par d'autres logiciels (bénéficiant éventuellement de sorties graphiques plus élaborées).

Pour chaque affichage, une valeur entière définit la fréquence avec laquelle il doit être remis à jour ; une valeur de 10 signifie ainsi que tous les 10 pas de temps, le programme re-dessinera l'image présentée à l'écran, ou écrira de nouvelles données dans le fichier concerné.

Pour le premier affichage, une seconde valeur entière définit également la fréquence avec laquelle de nouveaux points sont ajoutés à la courbe. Pour un point, la valeur ajoutée est en fait la moyenne d'une fonction sur un intervalle, défini par une troisième valeur entière, et qui peut-être réduit à un pas de temps. Moyenner une valeur sur un intervalle permet de lisser les variations trop importantes de certaines fonctions et d'observer le comportement moyen d'un indicateur. Envisageons par exemple le cas d'un indicateur très simple du succès des interactions, qui prend la valeur 1 lors d'une interaction réussie et 0 dans le cas contraire. Dans ce cas, un affichage brut des valeurs produit une courbe très difficile à lire, car oscillant trop rapidement entre les valeurs 0 et 1. Au contraire, moyenner les valeurs de l'indicateur sur un intervalle temporel suffisamment important permettra d'observer des variations moyennes significatives de l'évolution du succès des interactions.

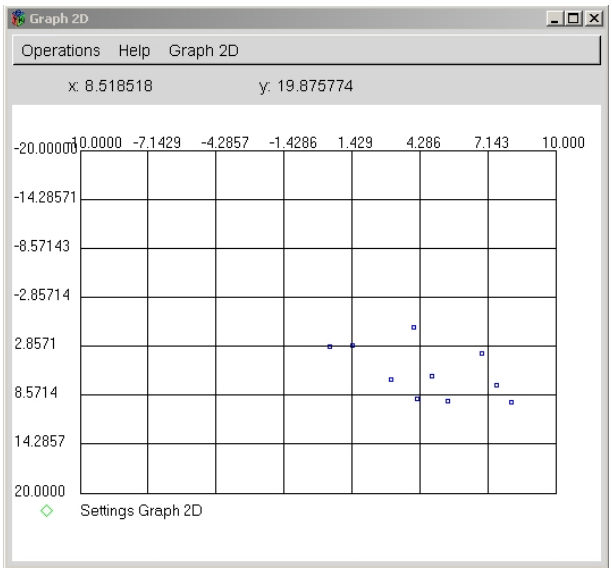

FIG. 3.13 – Le composant graphique **Graph2D**, pour la représentation d'espaces bidimensionnels

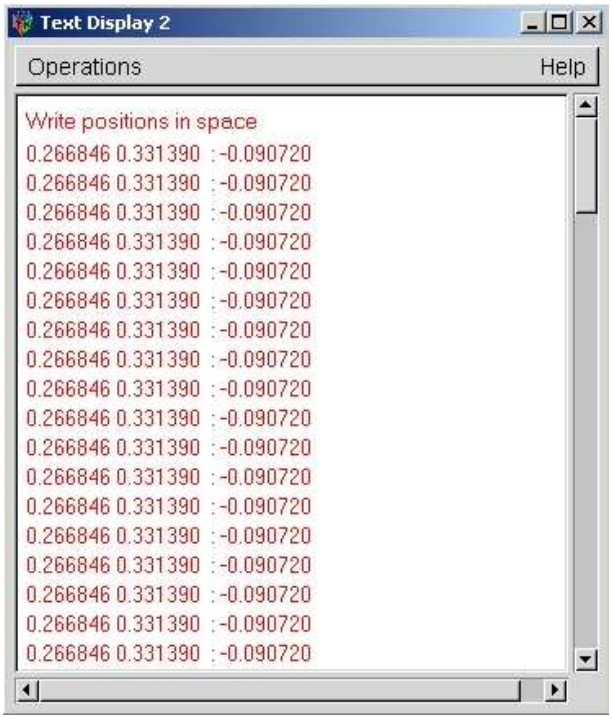

FIG. 3.14 – Le composant graphique **TextDisplay**, pour l'affichage de texte

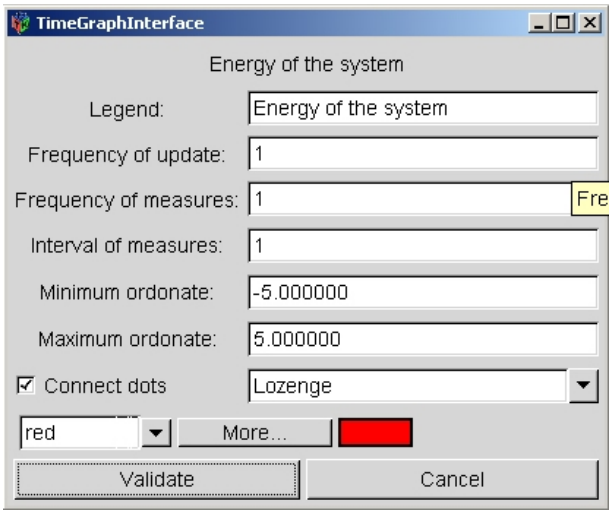

FIG. 3.15 – Une interface pour modifier les paramètres d'un prototype de courbe pour le composant graphique TimeGraph

Gestion des prototypes graphiques ou textuels. Nous avons introduit dans les paragraphes précédents les différents composants graphiques de l'interface. Pour pouvoir ajouter des éléments à ces composants (une courbe, un ensemble de points ou des lignes de texte), le programmeur va pouvoir définir des *prototypes*. Un prototype permet de définir les attributs de l'élément qui sera porté à l'écran dans l'un des affichages ou dans un fichier, par exemple pour les courbes :

- la couleur des points et des lignes qui les connectent éventuellement ;
- la figure (losange, cercle, croix...) identifiant la courbe dans la légende et sur le graphe ;
- les valeurs minimale et maximale pour l'axe des ordonnées ;
- l'interconnexion ou non des points (pour former une vraie courbe "brisée") ;
- les 3 valeurs entières de la fréquence d'affichage, de la fréquence d'ajout des points et de l'intervalle de moyennage pour établir les ordonnées de ces points ;
- éventuellement la valeur d'un argument qui sera transmis à la fonction indiquée par l'utilisateur pour le calcul de la valeur de la fonction au temps `t`.

Ce dernier argument permet de modifier le calcul de la fonction au cours de la simulation. Il pourra être remplacé par les paramètres que nous présenterons par la suite.

Il est nécessaire pour le programmeur de spécifier une première fois les prototypes dans le corps de son programme, en indiquant en particulier la fonction qui sera appelée pour obtenir les différents éléments à afficher. Cette fonction ne pourra plus être modifiée par la suite, mais l'utilisateur de l'interface pourra néanmoins modifier l'ensemble des autres attributs d'un prototype en cliquant sur sa légende dans le composant graphique correspondant. Un clic sur un élément entraînera l'apparition d'une interface semblable à celle présentée par la figure 3.15. Il est donc possible à chaque instant de modifier les graphiques pour obtenir en particulier les meilleures images à insérer dans des documents externes.

Paramètres et fonctions à répétition. Afin de rendre les simulations plus interactives, il est intéressant de pouvoir modifier certains paramètres du modèle au début ou en cours d'exécution. Ceci est possible grâce à l'objet **Parameter**, qui va lui aussi apparaître dans la liste d'éléments graphiques de la fenêtre principale (voir ci-dessous). Un clic de souris sur un paramètre affiche une fenêtre graphique contenant sa valeur, qu'il est alors possible de modifier.

Le dernier élément à apparaître dans la liste des prototypes sus-citée est le prototype de *fonction à répétition*. Cette dernière permet d'exécuter automatiquement avec une certaine fréquence et pendant un nombre de pas plus ou moins grand une fonction définie par l'utilisateur.

Génération de nombres aléatoires. Certaines simulations nécessitent de générer un nombre très important de nombres aléatoires, en particulier pour l'étude de mécanismes stochastiques. Le langage C++ dispose d'un générateur de nombres pseudo-aléatoires[29], mais il s'est révélé trop peu performant pour être utilisé dans les simulations que nous détaillerons par la suite, car le caractère non-aléatoire des suites de nombres générées venait fausser les résultats. Nous avons donc eu recours à un générateur baptisé générateur de Mersenne, qui offre une aléatoirité suffisante.

Description de l'interface graphique principale. Parallèlement aux différents affichages dans lesquels l'utilisateur peut faire apparaître différents types de données, une interface principale permet d'accéder aux principales fonctionnalités du logiciel, et de gérer les affichages précédents et les prototypes graphiques, comme le montre la figure 3.16.

Les menus dans la partie supérieure de l'interface offrent la possibilité de replacer une simulation dans son état initial (**reset**), de charger des scripts (voir ci-dessous), ou d'afficher des informations sur la plate-forme.

Les boutons à l'étage inférieur permettent le contrôle de l'évolution temporelle de la simulation. Un bouton permet l'initialisation de la simulation et un autre l'exécution d'un certain nombre de pas de temps. Ce nombre de pas est spécifié par l'utilisateur dans le cadre de droite.

La partie inférieure de l'interface est utilisée par une liste d'éléments qui vont permettre l'ouverture et le remplissage des composants graphiques introduits plus haut, ainsi que la gestion des paramètres et des fonctions à répétition. Un système d'icônes permet une identification rapide des différents affichages et des prototypes correspondants, ainsi que des paramètres.

Trois icônes permettent la création des composants graphiques **TimeGraph**, **TextDisplay** et **FileWriter** (sortie textuelle vers un fichier). Pour le composant **Graph2D**, chaque arrière-plan possible pour le graphique se traduit par une ligne dans la liste, tout comme les prototypes définis dans le programme de la simulation et les paramètres.

Afin de privilégier une utilisation intuitive et aisée grâce à la souris, le mécanisme requis pour la manipulation des graphiques et des prototypes est celui du *drag & drop* : après la création d'un affichage, l'utilisateur peut choisir l'un des prototypes en adéquation dans la liste par une pression sur le bouton de la souris, et le déplacer jusqu'à l'affichage pour l'y ajouter en relâchant le bouton. Ce mécanisme peut également fonctionner entre deux affichages de même nature, et l'utilisateur peut donc encore une fois régler au mieux ces affichages pour réaliser les images les plus pertinentes pour des documents externes.

[29] Il n'est pas possible pour un ordinateur (et encore moins pour un être humain) de générer des séries de nombres entièrement aléatoires.

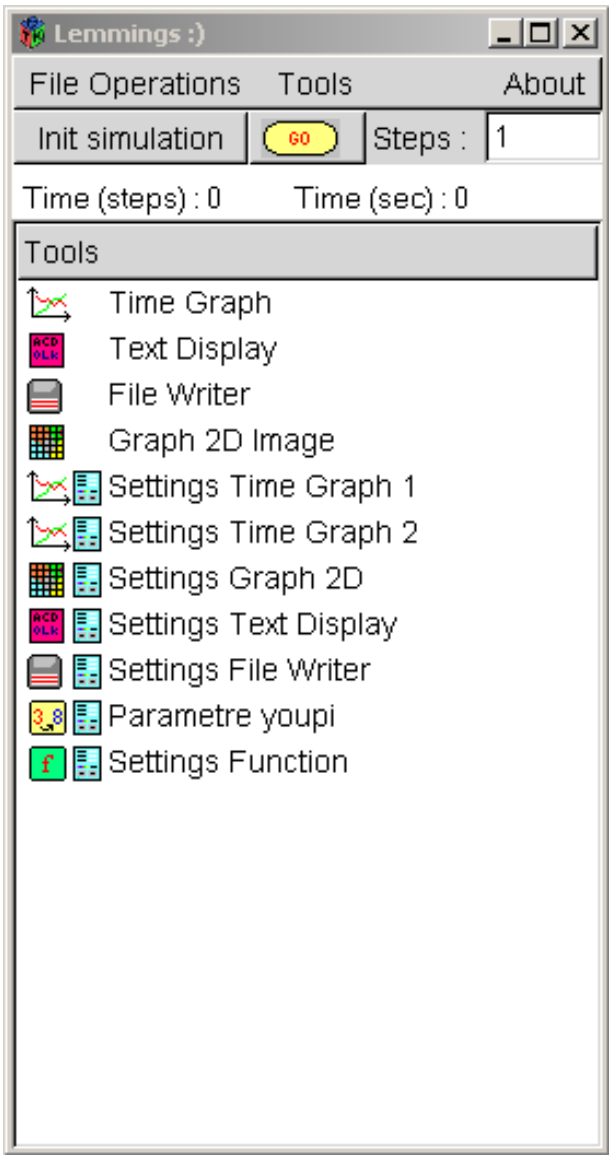

FIG. 3.16 – Interface graphique principale du logiciel Lemmings

Une poubelle située dans la partie inférieure de l'interface principale permet également d'effacer des représentations créées inutilement.

Scripts et utilisation "batch". Certaines validations de modèles ou expériences nécessitent l'exploration d'un espace de paramètres parfois important. Il suffit en effet de croiser les valeurs de 2 ou 3 paramètres sur des plages de valeurs même réduites pour obtenir un grand nombre de simulations à réaliser. Il est alors rapidement fastidieux de réaliser les mêmes opérations un grand nombre de fois pour la mesure des résultats des expériences (ouvrir les affichages, les remplir, effectuer des sauvegardes...). Si de plus le temps nécessité par une simulation est important, l'utilisateur est alors contraint de passer un temps considérable devant son écran pour contrôler le déroulement des événements.

Une réponse naturelle à ce problème est la possibilité de réaliser des traitements automatisés, où les différentes opérations à accomplir et les variations des paramètres sont prises en charge par la machine qui enchaîne les différentes simulations sans perte de temps. La spécification de ces chaînes d'action peut se faire par la lecture de fichier de *scripts*. C'est un tel mécanisme, bien que sous une forme assez rudimentaire, que nous avons introduit dans le noyau du logiciel LEMMingS.

Une suite de commande admissibles a été définie pour les fichiers de scripts, et permet l'ouverture, la fermeture et la sauvegarde des différents affichages, la modification des paramètres, l'exécution d'un certain nombre de pas de temps, la ré-initialisation de la simulation, ainsi que des boucles de type FOR permettant d'imbriquer les plages de valeurs des paramètres étudiés (ce qui permet de croiser ces derniers).

En conclusion, nous pouvons souligner le fait que la réalisation du noyau tel que nous l'avons décrit a fait appel à un certain nombre de techniques de programmation (dont certaines touchant aux limites du compilateur dont nous disposions (spécialisation partielle des classes *template* par exemple)) : fonctions *template* permettant de prendre en compte de façon générique des objets de types différents, utilisation de pointeurs sur fonctions, mise en place d'une interface graphique, et en particulier du mécanisme *drag and drop*... Le développement du logiciel a donc été un processus assez long, et a été en lui-même une source d'apprentissage et de progrès dans la maîtrise de la programmation.

Insertion du noyau dans les programmes de l'utilisateur. L'utilisation du noyau dans les simulations se fait de la façon suivante : l'ensemble des classes est défini dans une librairie que le programmeur intègre à la compilation dans son programme. Il suffit alors de faire démarrer l'application en "donnant la main" au noyau, c'est à dire en lui cédant la gestion des événements. Plus précisément, la bibliothèque graphique **Gtk+2.0** utilisée pour l'ensemble des composants graphiques est basée sur un mode de fonctionnement "événementiel". Après avoir été lancé, le programme se place dans un état d'attente. Lorsqu'il enregistre des événements (ouverture ou fermeture d'une fenêtre graphique, mouvements de la souris, click sur un bouton...), il invoque la fonction à laquelle l'événement est rattaché. Cette fonction peut alors déclencher une série d'actions, comme l'évolution de la simulation pour un certain nombre de pas de temps.

Description des modules

Pour les besoins des différentes simulations que nous avons développées lors de notre travail de recherche, différents modules touchant à différents domaines ont été conçus. Afin d'illustrer

cette diversité, nous pouvons préciser une partie de ces composants et introduire brièvement leurs fonctionnalités.

- une bibliothèque de fonctions mathématiques élémentaires absentes de la bibliothèque de base du C++ (**math.h**) : loi de Gauss et lois dérivées, distances euclidienne et dérivées, fonctions de traitement de tableaux (min, max, tri, produits de convolutions)... ;
- une bibliothèque pour la réalisation d'analyses spectrales de signaux sonores, à l'aide de transformées de Fourier. Ces analyses spectrales peuvent ensuite être visualisées grâce au graphe affichant des espaces bidimensionnels (voir figure 3.17) ;
- le modèle DRM (Distinctive Region Model) de Carré et Mrayati [Carré and Mrayati, 1993]. Ici, le code natif a juste été légèrement transformé et placé dans une bibliothèque. Ce modèle permet de simuler la production des voyelles (ainsi que de quelques consonnes) avec un tractus artificiel composé de tubes aux longueurs et sections variables ;
- un module regroupant différents outils pour le traitement des voyelles et la simulation de leur perception : transformations formants/barks - barks/formants, définition des principales voyelles... ;
- une base topographique terrestre et sous-marine de l'ensemble du globe, bâtie à partir des données de la base TerrainBase [Row and Hastings, 1999] des National Geophysical Data Center et World Data Center-A for Solid Earth Geophysics. Cette base fournit les altitudes pour l'ensemble de la planète avec une précision de 5 minutes d'angle (soit un point pour $25km^2$ environ). A l'aide d'un module graphique, il est possible de visualiser la topographie terrestre selon différentes projections cartographiques (fournies par le logiciel PROJ 4.3 [Evenden and Warmerdam, 2000]), en faisant des zooms sur les régions sujettes à l'étude, ainsi qu'en faisant varier le niveau marin. Ceci fut particulièrement utile pour le travail que nous détaillerons dans le chapitre 5. Il est également possible de faire évoluer des agents virtuels dans cet environnement (voir figure 3.18) ;
- un ensemble de réseaux de neurones artificiels tels le perceptron multi-couches, la carte de Kohonen ou le réseau hebbien auto-associatif. Un réseau de neurones beaucoup plus réaliste, inspiré du modèle de Hopfield et Brady a également été développé pour une étude sur le *perceptual magnet effect* en acquisition, qui rend compte de l'apparition des catégories perceptuelles (en collaboration avec Bong Au, du *Language Engineering Laboratory* de la *City University of Hong Kong*) [Hopfield and Brody, 2000] [Hopfield and Brody, 2001] ;
- un module pour simuler un univers bidimensionnel clos ou torique (avec les mesures de distance correspondantes, et des fonctions pour déplacer des agents dans cet environnement) ;
- des bibliothèques permettant la gestion de population d'agents (définition d'une classe "population", évolution démographique, contacts entre agents...) ;
- des modules "cognitifs" pour des agents plus sophistiqués, reposant par exemple sur des cartes de Kohonen modifiées pour l'acquisition de catégories vocaliques, des procédures d'interactions "linguistiques" entre agents...

Intégration des commandes du logiciel LEMMingS dans une simulation

Ce dernier paragraphe a pour but d'illustrer le procédé simple par lequel l'utilisateur inclut les fonctionnalités du logiciel LEMMingS dans le code de sa propre simulation. Le nombre de commandes permettant l'interconnexion avec le noyau est en fait relativement restreint. Nous

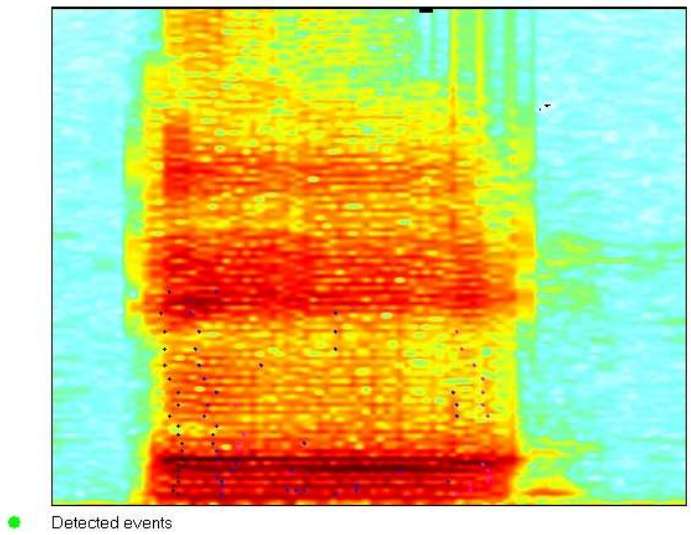

FIG. 3.17 – Exemple de module pour la plate-forme LEMMingS : transformée de Fourier et détection d'événements dans le signal

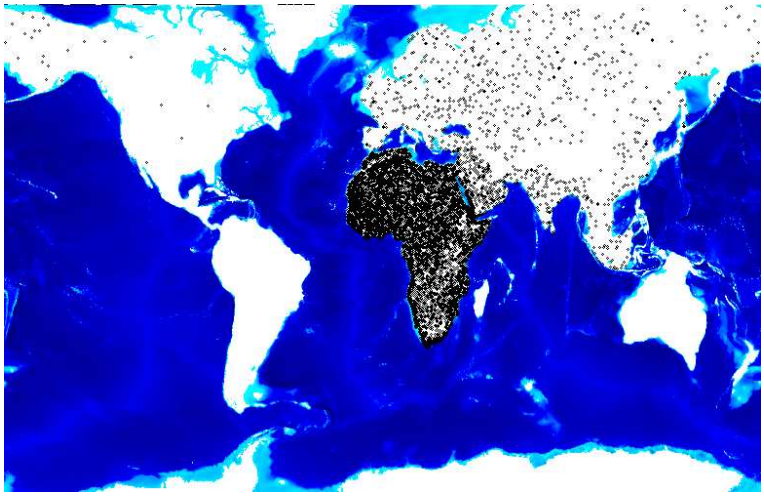

FIG. 3.18 – Exemple de module pour la plate-forme LEMMingS : module de cartographie

décrivons successivement les deux aspects majeurs de cette intégration, à savoir la définition de la classe principale du modèle, et le passage des prototypes graphiques.

Définition du monde de l'utilisateur. Afin de pouvoir mettre en œuvre les librairies du logiciel Lemmings, le code de l'utilisateur doit nécessairement reposer sur une classe dénommée **World**. C'est cette classe qui sera manipulée au niveau du noyau de LEMMingS. La définition minimale est spécifiée dans le code suivant :

```
#include "AllClasses.h"

class World : public AbstractWorld
{
public:

    void init();
    void creation();
    void evolution();
    void reset();

    World();
    virtual ~World();
};
```

La classe **Word** doit nécessairement contenir les quatres fonctions **init()**, **creation()**, **evolution()** et **reset()**. La première est celle appelée lors de l'initialisation du logiciel et de la simulation. La seconde est mise en œuvre lorsque l'utilisateur presse le bouton **Init simulation** de l'interface principale. La troisième sera appelée à chaque pas de temps de la simulation, et enfin la dernière permettra de réinitialiser une expérience (elle devra donc libérer la mémoire utilisée par les différents objets, remettre les variables à leur valeur initiale si nécessaire...).

L'intérêt de la fonction **init()** est de pouvoir envoyer des prototypes au noyau et les afficher à l'écran avant la création du monde de l'expérience avec le bouton **Init simulation**. Ceci permet de laisser à l'utilisateur la possibilité de modifier certains paramètres qui seront utilisés lors de la création du monde, par exemple les dimensions de l'environnement où se déplaceront des agents, le nombre d'agents...

Mise en oeuvre des différents composants graphiques et textuels. Les lignes de code ci-dessous illustrent l'inclusion de différents composants dans le programme de l'utilisateur.

Les déclarations des **Settings** seront effectuées dans les fichiers d'en-tête (d'extension **.h**). L'exemple suivant illustre la déclaration de deux **TimeGraphSettings**, d'un **FileWriterSettings** et d'un **TextDisplaySettings**, ainsi que celle des fonctions auxquelles ils réfèreront :

```
class World : public AbstractWorld
{
private:
...
```

```
    TimeGraphSettings *energy_time_graph;
    TimeGraphSettings *barycentres_variation_time_graph;
    FileWriterSettings *positions_file_writer;
    TextDisplaySettings *positions_text_display;

public:
...

    float get_energy();
    float get_barycentres_distance(int parameter);
    Vector<char *> *write_positions();
};
```

Les méthodes **get_energy()** et **get_barycentres_distance(int parameter)** correspondent aux deux **TimeGraphSettings**, tandis que la troisième méthode sera utilisée à la fois pour le **TextDisplaySettings** et le **FileWriterSettings**. Pour ce faire, elle renvoie un vecteur de chaînes de caractères, qui seront soit affichées à l'écran, soit inscrites dans un fichier.

Dans les fichiers de définition des méthodes (d'extension **.cpp**), une méthode contiendra la définition des différents prototypes, tandis que les différentes fonctions pour les **Settings** précédents seront définies de façon classique :

```
extern Command *command;

void World::creation()
{

    energy_time_graph =
        create_new_time_graph_settings<World>("Energy of the system",1, 1,
                                              1, -0.2, 0.05, true, red, square,
                                              get_energy, this);

    barycentres_variation_time_graph =
        create_new_time_graph_settings<World>("Evolution of the barycentre", 1, 1,
                                              1, 0.0, 5.0, true,  blue, lozenge,
                                               get_barycentres_distance, this);

    positions_file_writer =
        create_new_file_writer_settings<World>("Write positions in space", 1,
                                               write_positions, this);

    positions_text_display =
        create_new_text_display_settings<World>("Write positions in space", 1,
                                                green, write_positions, this);

    command->add_settings(energy_time_graph);
    command->add_settings(barycentres_variation_time_graph);
    command->add_settings(positions_file_writer);
    command->add_settings(positions_text_display);
...
}
```

```
float World::get_energy() {
    // Ici, il faut récupérer l'énergie...
    float *tab = population->get_items_frequencies_for_agent_and_context(0, 0);
    return population->get_energy(tab);
}

float World::get_barycentres_distance() {
    return population->get_barycentres_distance();
}

Vector<char *> *World::write_positions() {
    char *s;
    Vector<char *> *v = new Vector<char *>(101);
    float *tab;

    for (int i = 0; i < 100; i++)
    {
        tab = population->get_items_frequencies_for_agent_and_context(0, 0);
        s = (char *) malloc(50);
        sprintf(s, "%f %f  : %f", tab[0], tab[1], population->get_energy(tab));
        v->addElement(s);
    }
    return v;
}
```

La création d'un prototype de composant graphique ou textuel se fait en appelant la fonction correspondante : **create_new_time_graph_settings(...)** pour les **TimeGraphSettings**, **create_new_file_writer_settings<World>(...)** pour les **TimeDisplaySettings**...

Pour chaque fonction, un ensemble de paramètres spécifiques est transmis et permet de définir le cas échéant :

- la couleur des courbes ou des autres éléments graphiques (**TextDisplay**, **Graph2DSettings**, **TimeGraphSettings**) ;
- les figures des composants graphiques (**Graph2DSettings**, **TimeGraphSettings**) ;
- les valeurs minimale et maximale de l'axe des ordonnées (**TimeGraphSettings**) ;
- la fréquence et l'intervalle de mesure des données (pour les **TimeGraphSettings**).

En plus de ces paramètres spécifiques, d'autres paramètres sont communs à l'ensemble des définitions de prototypes : le nom du prototype tel qu'il apparaîtra dans l'interface principale et dans la légende des composants graphiques, la fréquence de mise à jour des composants à l'écran ou des fichiers, le nom de la fonction qui doit être appelée pour fournir les valeurs requises par le composant, et la référence à l'objet qui contient la fonction appelée, ici l'objet **world**, représenté par le pointeur réflexif **this**.

Une fois les prototypes définis, il suffit de les envoyer au noyau de LEMMingS pour qu'ils soient pris en compte et affichés à l'écran. Ceci se fait toujours par l'intermédiaire de la méthode **command→add_settings(...)**. L'objet **command** est l'un des deux objets principaux du noyau, défini dans d'autres fichiers, et ici utilisé grâce à sa déclaration externe (**extern Command *command**).

3.2.3 Conclusions

Nous pensons que la plate-forme LEMMingS peut être un outil performant pour le développement de simulations sur l'origine et l'évolution du langage et des langues. Le nombre de modules qui ont été développés au cours de la période 1999-2002 permet déjà d'aborder un nombre de situations variées, que ce soit à l'aide de réseaux de neurones, de systèmes multi-agents, en s'appuyant sur des données topographiques...

Un de nos soucis dans un futur proche est de proposer cette plate-forme à la communauté scientifique, en permettant l'accès non seulement au code du noyau et des modules, mais également en rédigeant un manuel d'utilisation complet et en documentant le code informatique de façon claire. L'utilisation du réseau Internet sera privilégiée pour faire connaître le logiciel, qui est déjà utilisé par un autre membre du laboratoire et en collaboration avec des étudiants de la *City University of Hong Kong*.

Premières conclusions et postulats d'étude

Il semblait impensable à père de sacrifier des millénaires d'évolution et d'industrie paléolithique, pour repartir à zéro en pauvres singes arboricoles. Notre grand-père, disait-il, se serait retourné dans sa tombe, laquelle se trouve à l'intérieur d'un crocodile, si son fils avait trahi tout l'effort de sa vie. Non, nous devions rester, et nous servir de notre tête. Il nous fallait trouver un truc pour empêcher les lions de nous manger, et une fois pour toutes. Mais lequel ? C'était le problème clé. Telle était la beauté de la pensée logique, disait-il : elle vous permet d'éliminer toutes les conjectures, jusqu'à ce qu'il ne reste que la dernière, qui est la bonne.

Roy Lewis, Pourquoi j'ai mangé mon père [Lewis, 1990] (p.40)

Il nous parait utile pour conclure cette première partie de définir précisément notre position et les postulats que nous avons adopté en conséquence dans la suite des études.

L'hypothèse la plus fondamentale que nous adoptons est celle d'une apparition du langage humain dans la continuité des systèmes de communication animaux, et un développement graduel de sa complexité, sans émergence brusque par le biais d'une macro-mutation génétique. Le langage ne bénéficie pas en outre de processus dédié ou de gène qui code de façon spécifique ses caractéristiques. Il s'inscrit dans des mécanismes cognitifs généraux, comme proposée par la linguistique cognitive.

Notre second postulat, à la suite des travaux de Dunbar, Dessalles etc., est que, parallèlement à sa capacité première de transport d'information, le langage est également un outil social, qui renforce les liens sociaux dans une communauté d'individus, mais permet aussi aux individus de se définir au sens large dans cette communauté, tout comme d'autres artéfacts culturels (mode vestimentaire, religions, comportements sociaux...). Nous adoptons l'argument de Dessalles en faveur d'une valeur localement optimale du langage, qui est pertinente vis à vis des théories évolutionnistes et nous semble en accord avec la dimension sociale que nous pensons être une caractéristique majeure du langage.

Dans le prolongement de ce second postulat, nous pensons que cette caractéristique sociale du langage s'est majoritairement préservée au cours du temps depuis le développement des premières formes de communication humaine jusqu'aux époques contemporaines. Contrairement à cette stabilité, les capacités cognitives et les structures démographiques ont évolué de façon

significative, et ont conduit aux langues contemporaines. De la même façon qu'une surenchère de complexification peut exister entre tricheurs et non-tricheurs chez certains animaux, le langage a pu se développer et *co-évoluer* avec les capacités cognitives de façon auto-catalytique par le renforcement de son rôle aux niveau social et également cognitif dans une seconde période. Son activité, tout en ayant une base sociale, a progressivement "envahi" l'ensemble de la cognition humaine, et a partiellement permis son développement.

Troisième postulat, le langage évolue sous l'influence de contraintes internes (physiologiques, cognitives, communicatives, sociales) et d'événements sociaux contingents. Cette évolution est *structurelle* et *structurellement déterminée*. Les langues constituent un système général assez proches de celui composé par les espèces biologiques, comme le pense Salikoko Mufwene. Il n'existe pas de rupture de transmission lors des phénomènes de contact linguistique, et les changements lors de ceux-ci, même s'ils sont parfois très significatifs, ne sont que les conséquences de phénomènes sociaux extrêmes et de réorganisations structurelles. Les phénomènes sociaux jouent le rôle de gâchettes pour les changements linguistiques : sans les déterminer (dans un sens qu'il reste toutefois à définir), ils participent néanmoins de l'évolution des langues.

Dernier postulat, il est possible en se basant sur la préservation du rôle social du langage, sur l'évolution des structures démographiques et des capacités cognitives d'émettre des hypothèses tangibles sur les langues de la préhistoire. Cette remontée dans le temps n'est peut-être pas permise par les techniques classiques de la linguistique historique, mais des modèles généraux pluridisciplinaires peuvent être pertinents et des approches typologiques peuvent mettre à jour des caractéristiques des langues actuelles héritées de très longue date. Les principes de la sociolinguistique peuvent être appliqués à des périodes reculées si l'on prend en compte les différences sociales et démographiques avec l'époque contemporaine (voir par exemple [Naccache, 2002] pour une proposition équivalente). Comme l'écrit encore William Labov :

"To draw further upon the parallel between language change and fashion change, it may be necessary to modify the uniformatarian principle, and consider that mechanisms of change may not be the same in other periods of history and other forms of society" [Labov, 2001] (p. 361)

A partir de ces différentes hypothèses, nous tenterons d'approcher quelques unes des questions importantes relatives à l'émergence des langues : comment les langues et la diversité linguistique sont-elles apparues ? A partir de quand peut-on parler de langues "modernes", c'est à dire de langues dont les caractéristiques typologiques sont essentiellement les mêmes que les langues actuelles ? Et pour finir comment les langues ont-elles évolué au cours de la préhistoire, et comment évoluent-elles aujourd'hui ?

Pour répondre à ces questions, nous aurons recours à plusieurs modèles informatiques, que ce soit pour étudier l'émergence de systèmes linguistiques ou leur évolution au cours du temps en fonction des différentes contraintes qui pèsent sur eux. Nos simulations seront toutefois assez éloignées de celles détaillées au chapitre 3 : en effet, notre soucis ne sera tout d'abord pas l'émergence de convention, mais l'évolution d'un système linguistique sous différentes contraintes naturelles et sociales.