Une approche générique de modélisation spatiale et temporelle : application à la modélisation de la dynamique des paysages

Pascal Degenne

Université Paris-Est

Thèse

présentée pour obtenir le grade de

Docteur de l'Université Paris-Est

Spécialité : **Informatique**

dans le cadre de l'Ecole Doctorale **MSTIC**

soutenue publiquement par

Pascal Degenne

le 13 mars 2012

Titre :

Une approche générique de la modélisation spatiale et temporelle : application à la modélisation des paysages

Directeur de thèse	**Robert Jeansoulin**
Encadrant	**Danny Lo Seen**

Jury :

Robert Jeansoulin	*Directeur de thèse*
Danny Lo Seen	*Encadrant*
Claude Monteil	*Rapporteur*
Eric Ramat	*Rapporteur*
Gilles Roussel	*Examinateur*
Alfred Stein	*Rapporteur*

Résumé

Les sciences qui traitent de la réalité, qu'elles soient naturelles, de la société ou de la vie, fonctionnent avec des modèles. Une partie de ces modèles décrivent les relations entre certaines grandeurs mesurables de la réalité, sans aller jusqu'au détail des interactions entre les éléments qui la composent. D'autres modèles décrivent ces interactions en prenant le point de vue des individus qui constituent le système, le comportement global n'est alors plus décrit à priori, mais observé à posteriori. Nous faisons le constat que dans les deux cas le scientifique a peu de liberté pour décrire les structures, en particulier spatiales, susceptibles de porter ces interactions.

Nous proposons une approche de modélisation que l'on peut situer à mi-chemin entre les deux, et qui incite à étudier un système à travers la nature de ses interactions et des structures de graphes qui peuvent les porter. En plaçant au même niveau les relations spatiales, fonctionnelles, sociales ou hiérarchiques, nous tentons aussi de nous affranchir des contraintes induites par le choix effectué souvent à priori d'une forme de représentation de l'espace.

Nous avons formalisé les concepts de base de cette approche, et ceux-ci ont constitué les éléments d'un langage métier, nommé Ocelet, que nous avons défini. Les outils permettant la mise en oeuvre de ce langage ont été développés et intégrés sous la forme d'un environnement de modélisation et de simulation. Enfin nous avons pu expérimenter notre nouvelle approche de modélisation et le langage Ocelet à travers la réalisation de plusieurs modèles présentant des situations variées de dynamiques paysagères.

Mots clés : Modélisation, Simulation, Graphes, Langage métier, Ocelet, Paysage

Summary

Sciences dealing with reality, be it related to nature, society or life, use models. Some of these models describe the relations that exist between measurable properties of that reality, without detailing the interactions between its components. Other models describe those interactions from the point of view of the individuals that form the system, in which case the overall behaviour is not defined a priori but observed a posteriori. In both cases, it can be noted that the scientist is often limited in its capacity to describe the structures, especially those spatial, which support the interactions.

We propose a modelling approach that can be considered intermediate, where the system is studied by examining the nature of the interactions involved and the graph structures needed to support them. By unifying the description of spatial, functional, social or hierarchical relationships, we attempt to lift constraints induced by the form of spatial representation that are often chosen a priori.

The basic concepts of this approach have been formalised, and were used to define and build a domain specific language, called Ocelet. The tools related to the implementation of the language have also been developed and assembled into an integrated modelling and simulation environment. It was then possible to experiment our new modelling approach and the Ocelet language by developing models for a variety of dynamic landscapes situations.

Keywords : Modelling, Simulation, Graphs, Domain Specific Language, Ocelet, Landscape

Remerciements

Le projet scientifique dont cette thèse est un élément est né d'une rencontre entre un praticien de la modélisation spatiale et un informaticien. Je joue dans ce binôme le rôle de l'informaticien, et Danny Lo Seen celui du modélisateur. Ainsi c'est d'abord grâce à toi Danny si j'ai pu me lancer dans ce projet de thèse et si il a été possible de le faire aboutir. Merci donc pour tout cela, mais aussi pour avoir été bien plus qu'un encadrant, un compagnon toujours prêt à partager de passionnantes discussions. Merci pour cette disponibilité et pour la curiosité scientifique dont tu fais preuve qui rend notre collaboration particulièrement motivante. Cette thèse est une étape, la meilleure façon de te remercier est je pense de continuer ensemble cette aventure, et j'en suis fort heureux.

Un grand merci à Robert Jeansoulin pour la confiance qu'il m'a accordé en acceptant de diriger cette thèse, et pour son soutien sans faille. Je reconnais pourtant que l'aventure dans laquelle nous nous sommes lancé, sur un sujet vraiment exploratoire, était plutôt risquée. Merci aussi Robert pour avoir répondu présent sans hésiter à chaque fois que j'en ai eu besoin, malgré notre éloignement géographique et ta charge de travail. Merci enfin pour la pertinence de tes conseils qui m'ont incité à réfléchir aux fondements scientifiques des idées que nous cherchions à développer.

Je remercie chaleureusement Claude Monteil, Eric Ramat, Gilles Roussel et Alfred Stein d'avoir accepté d'évaluer ces travaux, pour leur lecture attentive de ce document, pour le caractère constructif des critiques qu'ils ont apporté. Ces réflexions vont à l'évidence alimenter la poursuite de notre projet scientifique. Disposer d'un jury d'une telle qualité fut pour moi un grand honneur et un réel enrichissement.

Ces travaux n'auraient pu être réalisés sans les soutiens financiers de l'Agence Nationale pour la Recherche et du CIRAD. L'ANR a soutenu le projet blanc STAMP dont la problématique était au cœur de notre thèse. Le soutien du CIRAD s'est manifesté à travers la confiance qu'ont bien voulu nous accorder deux directeurs successifs de l'UMR TETIS : j'adresse ainsi mes remerciements à Pascal Kosuth et à Jean-Philippe Tonneau, grâce à qui ces travaux de thèse ont pu être pleinement intégrés dans mon activité professionnelle.

Le projet ANR STAMP a été l'occasion d'embarquer dans notre aventure quelques chercheurs de grande qualité. Je pense en premier lieu à Didier Parigot que j'ai retrouvé avec grand plaisir. Merci à toi Didier, d'abord pour tout ce que tu m'as appris dans cette discipline, mais aussi pour nous avoir incité à explorer de nouvelles voies à chacune de nos rencontres. Ton apport est inestimable, et je l'estime à sa juste valeur.

Je tiens à remercier chaleureusement Rémi Forax, pour toute l'aide qu'il nous a fourni dans la conception du langage Ocelet et la construction du compilateur, ainsi que pour la vivacité d'esprit dont il sait faire preuve qui a rendu nos réunions particulièrement créatives.

Merci à Olivier Curé pour le regard original qu'il a apporté à notre travail. Il nous a

montré comment orienter nos travaux vers les disciplines relatives à la représentation des connaissances. C'est une contribution importante et j'espère que nous serons à la hauteur pour explorer ces pistes plus avant.

J'adresse mes remerciements amicaux et enthousiastes à Christophe Proisy, capable de faire de la science de haut vol avec les pieds dans la vase. Il faut reconnaître que tes connaissances encyclopédiques de la mangrove, associées à un dopage à base de pains au chocolats, constituent un ensemble particulièrement efficace pour faire du bon travail ensemble. Merci pour ces paquets de vase que nous avons déplacé ensemble, et pour la mangrove que nous allons bientôt faire pousser dessus.

Merci à Cédric Gaucherel, un des rares chercheurs à avoir abordé très tôt les questions de modélisation de dynamiques spatiales avec des approches particulièrement originales, pour l'intérêt qu'il a porté à nos travaux. Je t'avais demandé d'avoir un oeil critique et jouer en quelque sorte le rôle de "poil à gratter", et tu as su enrichir nos réflexions sans pour autant gratter trop fort, de cela aussi je te remercie. J'en profite pour remercier Julie Alet, étudiante de Cédric, qui a gentiment accepté que l'on utilise les résultats de ses travaux pour construire l'un des modèles exposés dans cette thèse.

Merci enfin à chacun des autres participants au projet STAMP : Ayoub Ait Lahcen, Daniel Auclair, Christian Baron, Annelise Tran pour le soutien que vous nous avez apporté, pour nos échanges, et pour votre confiance.

Deux jeunes ingénieurs ont participé à cette aventure et je tiens à les remercier très sincèrement. Mathieu Castets a effectué un travail important sur le compilateur d'Ocelet et Rémi Tylski a fourni un travail tout aussi important sur l'environnement de développement. Merci pour le sérieux, et la qualité de vos réalisations qui nous ont permis de commencer à utiliser notre langage. Merci également à Valérie Vincent qui a mit ses talents de graphiste à notre service, c'est grâce à elle si Ocelet a un logo que nous sommes fier d'arborer.

Je voudrais adresser mes remerciements à Hugues Boussard de l'INRA de Rennes qui travaille sur des problématiques proches des nôtres. Nous avons eu la chance de nous rencontrer à plusieurs reprises et j'apprécie beaucoup son ouverture d'esprit et l'enthousiasme avec lequel il aborde les questions qui nous occupent. J'espère que nous aurons l'occasion d'avoir d'autres échanges tout aussi enrichissants à l'avenir.

Je souhaite remercier vivement Sylvie Cach, responsable administrative de l'école doctorale MSTIC à l'Université Paris-Est pour le soutient sympathique dont elle a fait preuve durant ces années de thèse. Merci également à Line Fonfrede, responsable administrative du Laboratoire d'Informatique de l'Institut Gaspard Monge, pour son aide précieuse lors de l'organisation de la soutenance de cette thèse.

Merci aux occupants de la Maison de la Télédétection pour leur soutien, leur qualité scientifique, leur amitié, et l'environnement particulièrement enrichissant que constitue

cet ensemble.

Bien sûr je veux remercier l'indispensable Jean-Claude, responsable de la reprographie de qualité à la MTD, pour ses conseils et les coups de mains lors de l'impression de cette thèse.

Merci Valé pour la joie de vivre que tu apportes tous les jours dans le bureau que nous partageons. Merci aussi de nous avoir fait découvrir les secrets de la vie des Aedes, des Culex, et de leurs mares. Je ne doute pas que nous aurons l'occasion de passer quelques autres vecteurs de virus dans notre moulinette à modéliser, et je m'en réjouis par avance.

Merci Valou pour tes encouragements Réunionnais fort bienvenus pour réchauffer un thésard cévenol en plein hiver. Merci aussi de me faire partager régulièrement tes qualités de photographe, cela m'a permis de voyager tout en rédigeant.

Je dois à mes parents de m'avoir transmis l'intérêt que je porte aux questions scientifiques, c'est un cadeau inestimable, et je les en remercie de tout mon cœur.

Enfin je voudrais surtout remercier Sabine, ma moitié, qui soutient mon travail dans toutes les circonstances avec une grande compréhension. Cette thèse n'existerait pas sans elle, c'est une évidence. Merci également à nos enfants d'avoir supporté un papa thésard, je ne le ferai pas une deuxième fois c'est promis.

Table des matières

Avant propos

De nombreux éléments de contexte ont influencés les travaux de cette thèse, du choix du sujet aux exemples d'application, en passant par les différents concepts qui ont été imaginés et mis en oeuvre. On ne peut tous les citer, mais nous voulons en présenter trois qu'il est possible d'identifier clairement : le CIRAD, l'UMR TETIS, et le projet STAMP.

Etant salarié du CIRAD depuis 2003, c'est dans ce contexte institutionnel que la présente thèse a vu le jour et a été réalisée. Le CIRAD est un établissement de recherche en agronomie pour le développement. Ses activités sont largement tournées vers les pays du sud avec une volonté affichée de favoriser les partenariats aux suds pour des travaux de recherche finalisés. Ces travaux s'articulent autour de six axes scientifiques prioritaires : 1) Intensification écologique, 2) Biomasse énergie, 3) Alimentation, 4) Santé animale et maladies émergentes, 5) Politiques publiques, et 6) Relations entre agriculture, environnement, nature et société.

C'est bien dans cette optique de recherche finalisée que nous avons orienté nos travaux, avec pour objectif la construction d'un outil de modélisation réellement opérationnel que nous pourrons utiliser et faire évoluer. Le contexte du CIRAD a bien sûr aussi orienté les cas d'applications que nous présentons dans cette thèse, et l'on peut d'ores et déjà présager que nombre des applications à venir trouveront leur place au moins dans les axes *Relations entre agriculture, environnement, nature et société* et *Santé animales et maladies émergentes*.

C'est aussi au sein de l'unité mixte de recherche TETIS (Territoire Environnement Télédétection et Information Spatiale) que nous avons travaillé. Il s'agit d'une unité pluridisciplinaire dont les travaux portent sur divers aspects de l'information spatiale à travers quatre axes de recherche :

- l'acquisition (axe ATTOS : Acquisition et Traitement de données de Télédétection et d'Observations Spatialisées)
- l'analyse et la modélisation (axe AMOS : Analyses et MOdélisations Spatiales), qui est l'axe principal dans lequel s'est inscrit notre travail de thèse.
- l'intégration au sein de systèmes d'information (axe SISO : Systèmes d'Information Spatialisée, modélisation, extraction et diffusion des dOnnées)
- son usage et son appropriation par les acteurs d'un territoire (axe USIG : Usage de l'Information Spatiale et Gouvernance)

La place privilégiée qu'occupe l'espace dans les travaux de cette unité est à l'origine de nos premières réflexions sur la façon dont celui-ci est pris en compte dans les modèles

de simulation. La nature essentiellement méthodologique des travaux effectués au sein de TETIS nous amène à échanger régulièrement avec des scientifiques de nombreuses disciplines différentes. Nous avons pu constater un besoin toujours croissant en matière de modélisation, avec souvent des difficultés pour intégrer la composante spatiale dans les modèles. C'est pour tenter de répondre à ces besoins, à travers une approche originale, que ce projet de thèse a été conçu et a été mené.

Enfin nos travaux se situent aussi dans le cadre d'un projet financé par l'Agence Nationale de la Recherche, qui a démarré en novembre 2007, pour une durée de 42 mois : STAMP (Modelling dynamic landscapes with Spatial, Temporal And Multi-scale Primitives)(Projet No. ANR-07-BLAN-0121), auquel nous avons activement participé et grâce auquel nous avons pu collaborer avec :

- Le laboratoire d'Informatique de l'Institut Gaspard Monge, de l'université Paris-Est à Marne-la-Vallée, en particulier Olivier Curé, Rémi Forax et Gilles Roussel.
- L'équipe Zénith de l'INRIA, basée à Montpellier et Sophia-Antipolis avec Didier Parigot et Ayoub Ait Lahcen.
- l'UMR AMAP à Montpellier, avec Christophe Proisy, Daniel Auclair, Cédric Gaucherel, Julie Alet.
- Des chercheurs du CIRAD : Christian Baron, Annelise Tran, Valérie Soti, Elodie Vintrou.

Ce financement a aussi rendu possible l'accueil temporaire de deux jeunes ingénieurs : Mathieu Castets et Rémi Tylski, qui ont contribué au développement d'un prototype d'outil de modélisation et de simulation.

Le travaux d'ingénierie réalisés pour rendre ce prototype opérationnel sont très importants, ils sont le fruit d'un travail collectif dans lequel j'ai joué les rôles de gestionnaire de projet et de développeur.

Cette thèse et le projet STAMP ont permis de jeter les bases, à la fois conceptuelles et opérationnelles, d'un projet de recherche dédié à la modélisation et la simulation de dynamiques spatiales et paysagères. Cela constitue une première étape qui sera suivie de plusieurs autres, pour traiter de certains aspects conceptuels que nous n'avons pas eu le temps d'explorer jusque là, mais aussi pour transformer notre prototype de plateforme de modélisation en un outil au niveau de finition suffisant pour être diffusé largement auprès de la communauté scientifique.

Ce document rend compte du travail de conception et d'expérimentation que j'ai effectué et qui constitue mon travail de thèse. Nous ne présentons que partiellement des travaux collectifs d'ingénierie qui ont été réalisés (dans le chapitre 5). Les expérimentations de modélisation, sur différentes problématiques, ne sont pas non plus présentées dans leur totalité. Nous avons en effet réalisé davantage de modèles que les trois que nous avons choisi d'exposer dans ce document. Et nous n'avons détaillé que certains aspects de ces modèles, notre propos étant ici d'illustrer l'usage de notre langage de modélisation et non les modèles eux mêmes. Certains de ces modèles ont été développés en collaboration étroite avec des collègues chercheurs et feront l'objet de publications particulières.

Ce document expose un point de vue particulier sur nos travaux, celui d'un informaticien, celui de la reflexion qui amène à la conceptualisation d'un langage métier et des éléments de sa réalisation.

Je vous souhaite bonne lecture.

Chapitre 1

Introduction générale

Sommaire

Les sciences qui traitent de la réalité, qu'elles soient naturelles, de la société ou de la vie, fonctionnent avec des modèles. Une partie de ces modèles décrivent le comportement global de certains aspects de la réalité, sans entrer dans le détail des interactions entre les éléments qui la composent. D'autres modèles prennent le point de vue des individus qui constituent le système. Les interactions sont régies par des règles qui sont portées par ces individus, et le comportement global n'est alors plus décrit à priori, mais observé à posteriori.

Nous faisons le constat que dans les deux cas le scientifique a peu de liberté pour décrire les structures susceptibles de porter ces interactions. Celles-ci ne sont en effet pas toujours réparties de manière unique : elles sont contraintes par des structures spatiales, hiérarchiques, sociales, ou fonctionnelles. Et ces contraintes devraient bien souvent être décrites à un niveau qui se situe entre le global et l'individu.

Notre objectif est de jeter un pont entre ces deux familles de modèles, et tenter d'améliorer notre capacité à intégrer dans ces modèles la compréhension que l'on a de l'organisation de ces niveaux intermédiaires.

La thèse que nous avançons dans ce mémoire repose sur un postulat : la structure qui porte les interactions dans un système n'est pas neutre vis à vis du comportement de ce système. Cela nous a amnené à aborder différentes questions sur la modélisation de ces structures :

- elle doivent pouvoir représenter ces différentes formes de contraintes que nous avons citées ;
- elles peuvent être dynamiques ;
- elles sont organisées, et cette organisation peut être un objet d'étude dans la mesure ou elle reflète certaines caractéristiques du système que l'on étudie.

C'est au travers de la conception d'un langage métier dédié à la modélisation et la simulation, et d'exemples d'applications qui ont inspirés nos travaux, que nous tenterons de démontrer notre thèse.

1.1 Enjeux de l'étude des dynamiques paysagères

L'augmentation de la population de la planète, les changements climatiques, l'émergence ou la réemergence de certaines maladies, la baisse prévisible des ressources en énergies fossiles, la perte de biodiversité engendrée par certaines activités humaines, sont des évolutions auxquelles nos sociétés doivent se préparer. La recherche a un important rôle à jouer dans cette préparation, d'abord par une évaluation quantitative de ces évolutions à travers de multiples formes de mesures, ensuite par une étude qualitative et cognitive pour essayer de comprendre les phénomènes et en estimer les conséquences, et enfin par la diffusion de ces connaissances pour aider à la prise de conscience et à l'éclairage de la prise de décision.

L'étude des paysages, de leur composition, de leurs structures spatiales et fonctionnelles, de leurs réactions face aux aléas climatiques, de leurs liens avec les activités humaines, mais aussi la modélisation de leur dynamique, sont des moyens de travailler sur ces questions, en particulier parce qu'ils combinent une prise en compte de l'espace, une vision systémique, et une approche nécessairement pluridisciplinaire. Ces trois aspects sont en effet au coeur de nombreux enjeux que nous venons d'exposer.

Comment adapter les pratiques agricoles pour réduire les usages de produits phytosanitaires, être plus économe en eau et en énergie, avec autant que possible une augmentation des rendements et des productions ? C'est à ce défi qu'est aujourd'hui confrontée la recherche en agronomie. L'une des voies explorées consiste à s'appuyer sur des processus écologiques de manière intensive, en remplaçant par exemple une partie des intrants minéraux par des produits organiques ou en associant plusieurs cultures en symbiose pour maintenir la fertilité des sols. Cela demande la gestion de tout un ensemble de pratiques sur un territoire (élevage, culture, co-construction de ces processus avec les acteurs locaux) qui nécessite une vision systémique, pluridisciplinaire et spatialisée des outils de production agricole.

Cette évolution de l'agronomie dépasse d'ailleurs l'aspect purement productif pour s'intégrer dans le cadre plus large de la gestion des espaces ruraux, avec en particulier une évaluation des systèmes de production par rapport à des services environnementaux auxquels ils peuvent contribuer.

En matière de santé animale on assiste à une progression de maladies émergentes ou reémergentes, qui affectent le bétail (comme la fièvre catarrhale ovine, ou la fièvre de la Vallée du Rift) dont certaines peuvent parfois se transmettre à l'homme (comme l'influenza aviaire). La caractérisation des paysages favorables au développement des vecteurs qui transmettent les virus, et la modélisation de la dynamique de ces paysages sont des outils qui peuvent aider à identifier les conditions favorable à l'émergence de grandes populations de ces vecteurs dans l'espace et dans le temps. De tels outils, couplés avec des modèles de déplacements des troupeaux et de leurs habitudes de rassemblement, pourraient aider à mieux appréhender les combinaisons de facteurs qui sont les plus propices à la diffusion de ces maladies et tenter d'anticiper les risques d'épizooties.

L'épidémiologie n'est pas le seul domaine dans lequel l'étude et la modélisation des paysages peut participer à la mise en place de systèmes d'alerte précoce. Les recherches en écologie du paysage ont montré à de nombreuses reprises à quel point le biotope de certaines les espèces vivantes est dépendant des structures spatiales du paysage, comme la position de ses constituants, les structures topologiques que cela forme, les possibilités de déplacement, les distances, les frontières ou encore le caractère plus ou moins fragmenté des écosystèmes qu'il abrite. L'étude de la dynamique de ces caractéristiques paysagères est essentielle pour évaluer les risques d'atteinte à la biodiversité. Leur modélisation permet de chercher des compromis entre développement économique et protection de l'environnement et de la biodiversité, dans l'accompagnement de projets de développement territoriaux.

Dans la plupart des domaines de recherche que nous venons de citer, la démarche expérimentale sur les paysages in vivo est particulièrement difficile et même souvent impossible. Nous sommes dans ce cas contraints de nous tourner vers la modélisation et la simulation pour tenter d'estimer les conséquences possibles de certains phénomènes naturels, de perturbations dues à l'activité humaine, ou de l'application de politiques agricoles.

1.2 Modélisation

1.2.1 Système

Le mot du grec ancien *système* qui signifie assemblage, ou « réunion en un corps de plusieurs choses ou parties », semble au départ proche du concept d'ensemble. Ce sens a évolué dans de nombreuses disciplines scientifiques pour contenir aussi une forme de structure entre les parties de cet ensemble. C'est par exemple avec ce sens qui inclut l'organisation des choses que Galilée emploie le terme dans sa comparaison des thèses géocentrique de Ptolémée et héliocentrique de Copernic [63] sur le mouvement des astres.

Avec l'industrialisation du XIX^{e} siecle vont être produites les premières études sur la régulation des machines, préparant ainsi le terrain pour un développement de travaux de recherche systématique sur la question de l'organisation et du contrôle d'un ensemble de parties en interaction les unes avec les autres.

Vers le milieu du XX^e siècle une importante activité scientifique s'est développée autour de la notion de système, et en particulier des systèmes *dynamiques*, c'est à dire dont les propriétés d'état ou de structure évoluent avec le temps.

Le biologiste K.L. von Bertalanffy, faisant remarquer que l'étude de la nature des interactions est chose commune dans de nombreux domaines de la recherche, propose la création d'une Théorie Générale des Systèmes [22, 145] dans laquelle il affirme qu'il faut prendre le système comme un tout : les éléments qui le constituent ne suffisent pas à expliquer ses propriétés, il faut aussi en comprendre l'organisation et les interactions entre ses éléments. Il appelle la communauté scientifique à théoriser les lois qui régissent ces interactions indépendamment de la discipline scientifique dans laquelle elles sont étudiées. Ce point de vue qui peut être considéré comme *holiste* s'oppose à la pensée *réductionniste* dominant jusque là la démarche scientifique.

Cybernétique et systémique

N. Wiener, A.Rosenblueth, R.Ashby, vont formaliser le mécanisme de régulation à l'aide du principe de rétroaction [13, 149], c'est à dire la façon dont un système peut se contrôler ou être contrôlé grâce à la circulation d'information entre les parties de ce système. Ils vont nous inviter à ignorer les détails de certaines structures et mécanismes d'un système pour les enfermer dans des boites noires et à considérer comment ces boites noires interagissent. Le concept d'interface se trouve mis en avant par rapport à la structure et aux mécanismes internes. Cette approche, à l'origine du terme *cybernetique* [149], va avoir une influence considérable sur la production scientifique de l'époque.

L'étude des systèmes en tant que tels, de leur dynamique, leur capacité à s'auto-organiser, les morphismes de systèmes que l'on peut retrouver dans des disciplines scientifiques aussi diverses que la biologie, la théorie de l'information, la sociologie, l'écologie, ou la psychologie, tend aujourd'hui à être regroupée sous le terme générique de *systémique* .

Sous l'impulsion de Heinz von Foerster, cette notion de rétro-action, c'est à dire d'auto référence d'un système va évoluer vers un changement de point de vue épistémologique de la cybernétique. Cette approche souvent appellée *cybernétique de second ordre*, ou *cybernétique de la cybernétique* implique le scientifique qui observe le système comme faisant partie prenante de l'environnement du système étudié, et l'étude du système ne saurait être complète sans la prise en compte de cet environnement.

Nos travaux s'inscrivent dans cette approche espitémologique ne serait-ce que par la subjectivité inévitable que le modélisateur apporte lorsqu'il construit le modèle d'un système étudié. Le sujet que nous traitons relève davantage de la création subjective que de la découverte.

Système ouvert, système fermé

Les termes *système ouvert* et *système fermé* concernent la relation qu'un système peut avoir avec son environnement. Pour chaque discipline scientifique il existe des définitions plus précises de la nature des relations considérées. En thermodynamique par exemple un système ouvert peut échanger de la matière avec son environnement, un système fermé ne peut pas échanger de matière mais peut quand même échanger de l'énergie. D'une manière générale dans ce document, nous parlerons de systèmes qui interagissent ou non les uns avec les autres ou avec leur environnement, la nature de ces interactions sera précisée selon les domaines d'applications considérés.

Si l'on considère un territoire, un paysage, ou un biotope en tant que système, il est difficile de l'imaginer comme étant isolé de son environnement. Pour le scientifique qui cherche à améliorer les connaissances d'un aspect particulier du monde, un effort d'abstraction est nécessaire. Il doit en effet chercher à isoler au moins partiellement certaines parties du système de leur environnement sans quoi il se retrouvera obligé de connaitre complètement cet environnement et l'environnement de cet environnement, jusqu'à la totalité du monde et de l'univers qui l'entoure. Il se retrouve contraint de faire appel à des abstractions qui sont plus proches du modèle représentant le système étudié que du système lui même. La notion de système fermé en particulier pourra être utilisée pour qualifier un système décrit (artificiellement) comme pouvant évoluer de lui même, et pour lequel on fait abstraction des échanges qu'il pourrait avoir avec son environnment.

Les systèmes étudiés sont souvent vus comme consitués de sous systèmes en interaction les uns avec les autres. On peut en général dégager des hiérarchies de systèmes de ces structures dans lesquelles chaque niveau englobant est considéré comme étant l'environnement des sous-systèmes dont il est constitué. Du point de vue d'un niveau englobant, tous les sous-systèmes dont il est constitué sont des système ouverts qui peuvent être influencés par leur environnement. C'est dans ce cas le système englobant le plus large, celui dont on aura délibérement fixé les limites, pour circonscrire le domaine d'étude, qui sera considéré comme fermé.

Système dynamique

On peut définir l'*état* d'un système comme un ensemble de valeurs que l'on mesure ou que l'on observe sur ce système à un moment donné, ou sur un intervalle de temps donné. On parle de *système dynamique* lorsque l'état du système évolue dans le temps.

Lorsque l'on est capable de décrire précisément la façon dont l'état d'un système peut évoluer, et que des causes identiques produisent les même effets sur cet état, on parle de *système déterministe*. Si en revanche l'évolution de l'état d'un système comporte une part d'aléatoire, et ne dépend donc pas uniquement des conditions initiales, on parle de *système stochastique*.

Il existe une troisième forme de système dynamique que l'on appelle *chaotique* : ce sont des systèmes pour lesquels il est impossible de prévoir l'état à un moment donné parce

que celui-ci évolue de manière infiniment complexe, mais les valeurs qu'il peut prendre appartiennent à un ensemble limite qu'il est parfois possible de caractériser et que l'on appelle un *attracteur*.

Systèmes complexes

"Un système est un système, pas un ensemble !" [93]. Par cette formule J-L Le Moigne exprimait avec concision le fait que certains systèmes portent une identité propre qui dépasse celles de leurs constituants. Cette identité peut parfois être appréhendée à travers l'observation et l'analyse de comportements singuliers. C'est par exemple la convergence d'un système dynamique vers un état stable, ou cyclique, que d'importantes variations de son environnement ne suffisent pas à empêcher. Cela peut aussi être l'apparition de comportements qu'il n'était pas possible de prévoir par l'analyse des parties qui le composent. Nous faisons référence dans ce dernier cas à la notion de comportement *émergent* qui est caractéristique de certains systèmes composés d'un grand nombre de parties en interactions.

Cette notion d'émergence dans les systèmes complexes peut être vue comme la source de l'auto-organisation. E. Morin considère d'ailleurs l'auto-organisation comme issue de la systémique et la cybernétique. Il en fait la source d'une réflexion éspitémologique entre sujet (qui porte une identité) et objet (élément dont est consitué le système) qu'il considère comme indissociables [108]. Par cette approche, E. Morin dépasse et complète les visions holistes et réductionnistes en affirmant que le système auto-organisé (sujet) et ses composants (objets) résultent d'une co-évolution. Celle-ci n'est pas le seul résultat d'interactions internes mais aussi du caractère ouvert du système, d'échanges permanents avec son environnement.

Ce n'est qu'avec l'avènement de l'informatique et de la disponibilité de capacités de calculs électroniques très importantes que cette discipline qui étudie les systèmes complexes a pu se développer. Notre incapacité à acquérir la connaissance de tels systèmes par le calcul analytique nous oblige en effet à faire appel à la simulation. Pour connaitre l'état du système à un moment donné, on est contraint de calculer les états successifs précédents. Cela revient à décrire des règles d'interactions élémentaires entre des éléments réprésentant des parties d'un système et à faire calculer par un ordinateur la façon dont le système évolue par étapes successives. On fait donc appel à des représentations abstraites d'un système : les modèles, et en particulier ceux dédiés à la simulation.

1.2.2 Modèle

Le terme *modèle* est suffisamment polysémique pour mériter d'être défini dans le contexte de notre travail et nous avons choisi de retenir deux définitions dont les mots clés font sens pour notre problématique :

1. Description d'un phénomène (ou d'un système), que l'on construit dans un but précis [91].
2. Toute structure qu'une personne peut utiliser pour simuler ou anticiper le comportement de quelque chose d'autre [105]

Dans la première définition nous voulons insister sur l'idée de *construction* d'une *description*, qui nous indique qu'un modèle est une forme de représentation que l'on fabrique intentionnellement. Lorsque nous allons parler de *modélisation* c'est bien à une activité de création que nous ferons référence. Ce premier aspect exprime la nature descriptive de la modélisation et l'on peut pour certaines problématiques s'en tenir à une modélisation descriptive.

La deuxième définition complète la première en précisant l'intention : *simuler* ou anticiper un *comportement*. Nos travaux sont en effet destinés à la construction de modèles destinés à la simulation de phénomènes que l'on cherche à connaître, à comprendre, à expliquer. Cet aspect prédictif de la modélisation est particulièrement utile pour étudier des systèmes pour lesquels la réalisation d'expériences sur le système lui même s'avère difficile, voire impossible. C'est le cas des études sur l'évolution des paysages, des systèmes écologiques et environnementaux qui constituent le contexte dans lequel nous travaillons.

"Modéliser c'est décider !"[93]. J-L Le Moigne fait remarquer à juste titre à quel point celui qui construit un modèle (le modélisateur) dispose d'une grande liberté créatrice. Il nous faut garder à l'esprit que pour un même objet ou système étudié, de nombreux modèles peuvent être conçus. Chacun dépendra de l'intention, de la finalité recherchée, voire même de l'expérience et l'état d'esprit du modélisateur. L'exercice de modélisation implique donc des choix sur le contenu du modèle lui même : "*décider des aspects du monde qui doivent être décrits dans le modèle et ceux qui doivent être délibérément ignorés*"[91].

Cet exercice implique aussi des choix sur les outils à utiliser pour construire un modèle. Cela peut aller de la représentation mentale du système considéré jusqu'au programme informatique permettant des simulations, en passant par le dessin, ou la formalisation mathématique.

1.2.3 Ingénierie du logiciel et méta-modèle

Dans certains cas, un modèle peut prendre la forme d'un logiciel, capable par exemple de simuler l'évolution de certaines caractéristiques du système étudié. Or un logiciel est lui même un système (un système de traitement de l'information), qui peut donc être lui aussi modélisé.

Ainsi, en ingénierie des logiciels le terme *modèle* désigne une représentation formelle de l'ensemble des concepts présents dans un logiciel et des relations qui existent entre ces concepts. Cette représentation est destinée d'une part à favoriser la compréhension et les échanges entre les différents intervenants liés à un projet de développement de système d'information, et d'autre part à permettre la projection de ce modèle sur différentes plateformes d'implémentation.

Les travaux relatifs à l'ingénierie des systèmes d'information ont permis la création d'un certain nombre de standards en matière de modélisation dont le plus connu aujourd'hui est certainement le langage UML (Unified Modeling Langage)[7].

On voit que l'on peut penser la modélisation à plusieurs niveaux et l'on désigne généralement le modèle décrivant la façon même de modéliser par le terme *méta-modèle* [6]. L'usage de méta-modèle doit permettre la mise au point d'outils de transformation de modèle. Ces transformations peuvent servir par exemple à conserver la cohérence entre un modèle construit à l'aide d'un éditeur graphique et la projection de ce modèle dans un langage de programmation particulier, sur une plateforme particulière.

Considérer qu'un modèle est lui-même susceptible d'être modélisé permet donc d'automatiser un certain nombre de tâches liées à la modélisation des systèmes d'information. L'exercice de modélisation lui même reste malgré tout dépendant de l'intention de celui qui modélise. Et les méthodes à même d'aider à faire les bons choix en matière de modélisation (ou de méta-modélisation) sont à ce jour des questions ouvertes [112].

1.3 Points de vue sur le paysage

Le paysage peut être considéré comme une partie de territoire que l'on observe, comme un espace que l'on peut façonner, et gérer, ou encore comme un phénomène qu'il est possible d'étudier avec méthode, dans sa structure, ses interactions, sa dynamique ; dans tous les cas le paysage est indissociable d'un certain point de vue, implicite ou délibéré.

Le point de vue du peintre, ou du photographe sera choisi pour favoriser la composition, la lumière, le cadrage, un ensemble de facteurs dédiés à l'image, la représentation picturale, la capture de l'instant et d'une émotion.

Le point de vue de l'architecte et du paysagiste sera sans doute plus technique, avec un souci d'intégration du paysage construit dans le cadre environnant, et aussi une vision davantage dynamique de la façon dont ce paysage va évoluer dans le temps.

Le scientifique s'intéressera plutôt au paysage-phénomène, qui se prête à l'analyse méthodologique, rigoureuse, mais qui n'échappe pourtant pas à la nécessité d'un point du vue : celui de l'objectif recherché par cette analyse et celui de sa discipline scientifique. Le paysage étudié par un géographe, à travers par exemple les interactions entre société et territoire, a toutes les chances d'être fort différent du paysage étudié par un écologue ou un par un hydrologue en un même lieu. Chaque discipline distinguera des éléments relatifs à sa propre problématique, qu'il s'agisse des constituants dotés d'une extension spatiale ou des flux de matière ou d'information entre ces constituants.

Au delà de l'analyse, c'est la construction de modèles de paysages et de leur dynamique qui fait l'objet de nos travaux. Or, nous venons de le voir, l'exercice de modélisation présente un aspect délibérément créatif, ce qui implique le choix conscient et si possible argumenté d'un point de vue particulier sur le paysage en tant que système. Ce choix dépendra de la discipline scientifique mais aussi et surtout de l'intention à l'origine de la modélisation : que cherche-t-on à simuler ? et dans quel but ? Cette intention est propre à chaque projet, à chaque étude, et nous devons essayer de rendre nos outils de modélisation aussi adaptables que possible pour qu'il y ait adéquation entre l'intention et les moyens. Nous tentons dans ce qui suit de donner des éléments de ce qui signifie le paysage pour certaines disciplines relatives au contexte thématique de nos travaux, et d'en tirer des caractéristiques importantes pour la conception de modèles.

En géographie, c'est après avoir dépassé l'aspect purement descriptif du paysage dans la deuxième moitié du XX^{eme} siècle que des travaux vont montrer d'abord l'intérêt de la prise en compte de l'action humaine dans l'étude du paysage, avant d'évoluer vers une approche plus systémique intégrant de nombreux aspects dans une vision dynamique [35]. On peut citer en particulier les concepts de *Géosystème* proposé par N. Beroutchachvili et G. Bertrand ou le *Modèle systémique paysage* proposé par Th. Brossard et J.C. Wieber qui sont deux exemples datant de 1978 et 1984 de travaux relatifs à la vision systémique du concept de paysage [54]. Les travaux de R. Brunet [28] ont apporté une dimension proche de la modélisation à travers les *chorèmes*[1] : une forme schématique de représentation de structures spatiales, de dynamiques et d'interactions sur un territoire.

Nous retiendrons que pour le géographe, le paysage est le résultat d'interactions entre espace et société. L'espace n'est pas seulement un support organisé, il est aussi organisant [144]. Nous devons prendre en compte les relations entre dynamiques anthropiques, dynamiques environnementales, et espace [143]. Ces trois aspects de la modélisation sont complémentaires et forment un système rétro-actif : la société agit sur l'organisation de l'espace, et en retour la structure et l'organisation de l'espace établit de nouveaux rapports entre le paysage et la société. Si l'on veut modéliser des dynamiques paysagères selon ce point de vue, il nous faut pouvoir représenter la diversité de ces formes d'interactions : à la fois spatiales, structurelles et sociales.

G. Bertrand insistait en 2002 sur le fait que la géographie n'est plus seule à étudier le paysage dans sa structure et dans son fonctionnement [23]. D'autres disciplines en effet ont pris conscience de la nécessité d'intégrer la notion de paysage, en tant que système dans leur problématique. Il s'agit principalement de disciplines qui pendant longtemps se sont intéressées à des variables agrégées et qui ont analysé les lois générales qui contraignent l'évolution temporelle de ces variables sans prendre en compte leur dimension spatiale.

C'est en particulier le cas de l'écologie dont une branche, *l'écologie du paysage*, s'est orientée il y a une trentaine d'années vers une étude systématique des relations entre les espèces vivantes et les structures spatiales des milieux dans lesquelles elles vivent. Ces travaux ont mené à une vision du paysage en tant qu'objet faisant partie prenante des systèmes écologiques. Le paysage possède, selon ce point de vue, des structures et des fonctions.

Les structures sont constituées de communautés ou d'espèces formant des *taches*[2] possédant des caractères homogènes qu'il est possible de distinguer de leur environnement. Ces *taches* présentent des caractéristiques spatiales mesurables telles que la taille, la forme, leur nombre, leur distribution dans un environnement [56]. De nombreux travaux ont été réalisés sur les mesures de ces caractéristiques spatiales [80, 139] et l'on peut noter l'existence d'un logiciel (nommé Fragstat) dédié à ces types de mesures [100]. Un résultat important de ces travaux est le constat que l'hétérogénéité mesurée, et la connectivité des

[1] Les chorèmes aident à une modélisation dans le sens de la représentation des connaissances et non de la simulation. Ils peuvent probablement trouver leur place dans les phases d'analyse et de conception de modèles de simulation.

[2] C'est le mot anglais *patches* qui est généralement utilisé et qui se retrouve traduit de diverses façons dans la littérature en français.

taches sont fonction de l'échelle d'observation avec des effets de seuils qu'il est possible de quantifier.

Les fonctions correspondent aux types d'interactions observables entre les structures du paysage (taches, corridors, frontières, etc.) et processus écologiques. Les corridors facilitent par exemple les déplacements des certaines espèces d'un milieu favorable à un autre. Les frontières agissent sur les échanges entre l'intérieur des *taches* et leur environnement, elles peuvent avoir un effet de filtre pour certaines espèces. Parmi ces interactions il faut aussi prendre en compte les perturbations que certaines espèces peuvent provoquer par leur action sur le paysage, et en particulier celles dues aux activités humaines. Toutes ces fonctions sont à la base de ce qui permet d'analyser, de comprendre, et de modéliser les dynamiques écopaysagères.

Certains des travaux sur les liens entre structures spatiales et processus écologiques se sont appuyés sur la construction et l'analyse de *modèles neutres* [67]. Ce sont des modèles qui (par opposition aux modèles *explicites*) sont construits de manière totalement artificielle pour étudier les structures produites par un mécanisme considéré de manière isolé (comme la percolation, la construction de structures fractales, différentes formes de distributions aléatoires etc.), sans aucune implication de données ou de processus réels. La connaissance des caractéristiques propres aux modèles neutres permet une meilleure identification et caractérisation de structures observées dans des modèles explicites. Les modèles neutres ont par exemple montré leur intérêt pour le développement d'indices spatiaux de structures paysagères, la prédiction de seuils critiques, la caractérisation du concept de connectivité dans les paysages, l'étude de la perception propre qu'une espèce peut avoir des structures du paysage, ou encore la détermination de l'influence de l'hétérogénéité spatiale sur les processus écologiques [151].

La recherche en écologie du paysage a fait évoluer la compréhension des causes et conséquences des hétérogénéités spatiales, leur dépendence vis à vis de l'échelle d'étude, et a influencé les pratiques en matière de gestion du territoire. Les liens entre structuration du paysage et processus écologiques restent depuis longtemps au centre des questions de recherche en écologie du paysage [29, 138], et les modèles de simulation sont des outils privilégiés pour étudier ces questions.

L'écologie du paysage met donc l'accent sur les interactions spatialisées dans les systèmes écologiques. Nous retrouvons la nécessité de prendre en compte des relations de diverses natures et de les combiner. Nous retenons aussi l'importance de l'échelle pour l'analyse et la modélisation des structures spatiales, ainsi que des liens entre processus décrits à différentes échelles. La connectivité des habitats, les corridors, les frontières (en tant que limite ou en tant que filtre), sont des aspects importants à prendre en compte pour la modélisation des paysages. Les réseaux de circulation que ces structures constituent sont en effet d'un haut intérêt pour les enjeux de conservation des espèces [36, 104, 141]. Notons que l'on trouve ici aussi cet effet retro-actif des dynamiques paysagères : la répartition des espèces contribue à structurer le paysage, et les structures spatiales qui en découlent agissent à leur tour sur la distribution des espèces.

La géographie et l'écologie du paysage sont sans doute les deux domaines dans lesquels

les dynamiques paysagères ont été le plus étudiées. Bien d'autres domaines sont cependant concernés par le sujet, et même si le paysage n'est pas au centre de leurs problématiques, les besoins en outils d'analyse, de modélisation et de simulation sont très importants. C'est le cas par exemple des dynamiques forestières, des dynamiques côtières, de diverses questions relatives à la biodiversité, et de manière plus générale de nombreuses questions environnementales comme les conséquences de l'évolution du climat sur l'agriculture.

1.4 Une approche générique de la modélisation spatiale et temporelle : objectifs de la thèse

Les outils de modélisation et de simulation dont il est question dans cette thèse sont destinés à des applications de recherche finalisée, pour lesquelles il est nécessaire de combiner une approche spatiale a une vision systémique et pluridisciplinaire d'une problématique de recherche. La modélisation de dynamiques paysagères pose des questions de recherche qui combinent ces différents aspects et constitue notre principal domaine d'expérimentation.

L'objectif général de cette thèse est d'assurer à de tels modèles, une indépendance vis-à-vis des formalismes utilisés pour la représentation de l'information géographique.

Notre hypothèse principale est que les modèles sont trop souvent dépendants d'une forme de représentation de l'espace choisie a priori, et que cela contraint l'exercice de modélisation et limite les capacités d'expression du modélisateur. Il est en effet communément admis que l'espace peut être modélisé soit par une forme discrétisée de champs de valeurs continus, généralement sous la forme d'une grille dont on fixe une fois pour toute la taille des cellules, soit par des objets clairement délimités, construits à base de points, ligne, surfaces, ou volumes, et réunis en couches thématiques (c'est le mode de représentation que l'on trouve dans la plupart des systèmes d'information géographiques).

La première forme de représentation est adaptée à la modélisation de dynamiques d'états, mais ne permet pas de faire évoluer la topologie des voisinages, ni de travailler à plusieurs échelles différentes.

La seconde offre davantage de possibilités en matière de représentation de structures spatiale et est particulièrement bien adaptée à l'analyse spatiale et à la représentation cartographique. La dynamique des structures spatiales reste par contre difficile à modéliser, en particulier pour les objets qui doivent changer de forme, se diviser, ou dont les limites ne sont pas clairement définies.

Nous nous sommes donné deux sous objectifs qui constituent des moyens visant à atteindre notre objectif général :

1. Proposer une forme générique de représentation des éléments et interactions d'un système, qui permette d'intégrer dans un modèle ses aspects spatiaux, systémiques et pluridisciplinaires et d'en décrire la dynamique.
2. Proposer un outil de simulation basé sur cette forme de modélisation. Un tel outil

doit permettre la construction d'un état initial, la description de règles qui régissent la façon dont le système peut évoluer et l'exécution d'expériences de simulation.

1.5 Structure du document

Si nous avons entamé un travail de recherche relatif aux méthodes et outils de modélisation et de simulation pour les sciences de l'environnement, c'est que nous avions l'intuition que les possibilités offertes par l'état de l'art ne répondent que partiellement aux besoins de ce domaine. Dans le **chapitre 2** cette intuition est mise à l'épreuve d'une analyse des paradigmes, formalismes et outils de modélisation à la disposition des chercheurs en sciences de l'environnement. Nous portons un regard attentif à la façon dont l'espace est pris en compte, c'est à dire sous quelle forme il est représenté dans les modèles, et quelles contraintes peuvent être associées à ces formes de représentation.

Nous pensons en effet que les structures spatiales qui induisent certaines formes d'interactions ne sont pas neutres dans l'expression de la dynamique d'un système, et notamment dans le cas des dynamiques paysagères. Nous proposons de placer ces structures spatiales au même niveau que d'autres formes de structures : hiérarchiques, fonctionnelles, sociales par exemple.

Nous tirons de cette étude des enseignements qui orientent nos choix vers une approche originale de modélisation dans laquelle le concept de graphe va jouer un rôle central.

L'approche de modélisation que nous proposons est développée dans le **chapitre 3**. Nous abordons, en différentes étapes, les concepts sur lesquels nous allons nous appuyer, et nous en donnons des définitions formelles. L'importance des graphes (en particulier ceux que nous nommons *graphes d'interaction*) dans cette approche nous incite aussi à aborder la question de leur construction.

Dans la deuxième partie de ce chapitre ce sont donc les liens entre sources de données, notamment les données géographiques, et structures de graphes que nous examinons. Cette étude nous aide à préciser la nature des outils à construire pour établir ces liens et les faire évoluer dans un modèle.

Ce chapitre essentiellement conceptuel répond au premier sous objectif que nous nous sommes donné et prépare le second : la mise au point d'un outil basé sur ces concepts.

Nous avons choisi de proposer un outil de modélisation sous la forme d'un langage métier qui se nomme *Ocelet*. Nous argumentons ce choix en introduction du **chapitre 4** qui est consacré à la description détaillée de ce langage. Nous en décrivons chacun des éléments, parmi lesquels on retrouve les principaux concepts décrits dans le chapitre précédent.

Le **chapitre 5** donne un aperçu des travaux d'ingénierie réalisés pour donner vie au langage Ocelet. Ce sont d'abord le compilateur et le générateur de code, qui permettent

la traduction des modèles écrits avec Ocelet en langage Java. C'est aussi le moteur d'éxécution qui est un ensemble de classes Java que nous avons écrites pour assurer la partie générique de la sémantique du langage. C'est enfin un environnement de modélisation, qui intègre l'ensemble des outils, et offre un cadre de travail homogène dédié à la réalisation de modèles et aux expériences de simulation.

La conception de notre approche de modélisation ne s'est pas faite de manière linéaire. Nous avons effectué en permanence un va et vient entre une vision asbtraite et conceptuelle de nos travaux d'une part, et des situations concrètes de modélisation d'autre part. Certains de ces exercices de modélisation présentent des particularités à même d'illustrer des aspects importants de notre approche. Trois modèles ont été retenus dans cet esprit et sont exposés dans le **chapitre 6**. Leur examen peut donner un éclairage complémentaire sur Ocelet et sur la façon de raisonner lorsque l'on modélise à l'aide de graphes d'interaction.

Enfin le projet de recherche que nous avons initié avec cette thèse va se poursuivre au delà des travaux présentés ici. Nous discutons en **Conclusion** quelques unes des perspectives envigasées.

Chapitre 2

Les moyens de la modélisation en sciences de l'environnement

Sommaire

2.1 Introduction

Pour qui souhaite construire un modèle à des fins de simulation se pose la question du choix d'un formalisme de modélisation. Celui-ci doit permettre d'exprimer correctement les concepts que l'on cherche à intégrer et manipuler dans le modèle. En sciences de l'environnement, on doit bien souvent intégrer des éléments de natures différentes et le choix du meilleur formalisme va rarement de soi.

Nous présentons dans le présent chapitre un aperçu des différentes approches disponibles, les paradigmes sur lesquels elles sont fondées, et les contraintes qu'elles peuvent porter.

On pourra noter à travers cette présentation que la diversité des approches est le résultat d'une évolution historique, en partie liée aux progrès techniques en matière de calcul électronique. Ces progrès ont permis d'obtenir des solutions approchées de problèmes

jusque là inaccessibles à une résolution analytique, et ont donné accès à la manipulation de très grandes quantités d'éléments dans un modèle.

Cette évolution est aussi liée à la disponibilité de nouvelles formes d'acquisition et de traitement de données, comme les mesures de télédétection et les outils d'analyse spatiale. La capacité à disposer de grande quantités de données géoréférencées a fait naître de nouveaux besoins dans les domaines de la représentation des connaissances, de la modélisation et de la simulation.

Ces besoins et techniques continuent d'évoluer aujourd'hui, les travaux de la présente thèse peuvent être vus comme une tentative d'accompagnement de ces évolutions, et nous verrons en fin de chapitre où se situe dans ce contexte l'approche que nous proposons.

2.2 Des modèles analytiques à la dynamique des systèmes

On cherche à exprimer les relations entre des variables représentant des grandeurs caractéristiques d'un système. En mathématiques, les équations différentielles constituent un outil privilégié pour l'étude de telles relations. Cet outil formel permet d'étudier de manière analytique ou géométrique les domaines de valeurs qui constituent des solutions pour ces équations. Pour les systèmes dynamiques, on cherche en particulier à déduire de ces équations les possibles évolutions temporelles des variables comme la convergence vers un équilibre ou le caractère cyclique de leurs variations.

On a par exemple fait appel très tôt aux équations différentielles pour l'étude de la *dynamique des populations*. En 1798, T. Malthus remarquait [97], que la croissance d'une population suit une loi exponentielle alors que les ressources dont elle dépend suivent une croissance arithmétique. Ce principe d'évolution d'une population fut mis en équation par Verhulst en 1838 qui proposa l'équation suivante [142] :

$$\frac{dp}{dt} = mp - \varphi(p)$$

p étant le nombre d'individus dans une population, la partie mp exprime la croissance exponentielle d'une population et la partie $\varphi(p)$ représente une pression proportionnellement croissante sur cette population due aux limites de ressources en espace ou en nourriture.

Cette équation que l'on appelle *logistique* a été depuis abondamment étudiée et adaptée dans de très nombreux modèles portant sur la croissance d'une population seule. Sa version moderne s'exprime par [21] :

$$\frac{dN}{dt} = aN(1 - \frac{N}{k})$$

où N est la densité de biomasse de la population (entre 0 et 1), a est sa capacité maximale de croissance, et k est la densité d'équilibre de cette population. Malgré son apparente simplicité, cette équation peut converger vers une valeur stable, donner des oscillations, ou montrer un comportement chaotique selon les valeurs de a.

En épidémiologie, c'est à R. Ross que l'on doit les premiers travaux de modélisation mathématique. Il proposa en 1911 un *modèle à compartiments* en annexe de la deuxième édition de son ouvrage sur la prévention du paludisme [131]. Ce travail lui permit de montrer qu'il est théoriquement possible d'éradiquer le paludisme en faisant baisser la population des vecteurs de transmission (les anophèles femelles) en dessous d'un certain seuil.

En 1927, W. O. Kermack et A. G. MacKendrick publièrent une étude formelle sur les modèles à compartiments [89] qui contribua largement à leur donner la forme que l'on utilise aujourd'hui[43] : ces modèles sont construits à partir d'une partition d'une ou plusieurs populations. Chaque partie constitue un *compartiment* qui correspond à l'état d'un ensemble d'individus. Les équations du modèle décrivent à quelle vitesse les individus peuvent changer de compartiment.

Le modèle de Ross par exemple comportait deux compartiments : les susceptibles et les infectieux. Le modèle décrit par Kermack et MacKendrick est aujourd'hui appelé : S I R (Susceptible, Infected, Recovered). Une population est donc divisée en trois compartiments et la recherche analytique de solutions est plus ou moins complexe selon les équations que l'on choisit pour décrire les modalités de passage d'un compartiment à l'autre.

Sensiblement à la même époque (1925-1928), ce sont les interactions conjointes entre deux populations qui étaient mises en équation par Lotka et Volterra [113] dans leur *modèle proie-prédateur*. Il s'agit d'un couple d'équations différentielles exprimant les variations de l'effectif d'une population de proies, notée x, et d'une population de prédateurs, notée y :

$$\begin{cases} \frac{dx}{dt} = x.(\alpha - \beta.y) \\ \frac{dy}{dt} = y.(\delta.x - \gamma) \end{cases}$$

Pour la population de proies, α représente le taux de naissances et β le taux de mortalité du à la prédation. Pour la population de prédateurs, δ est le taux d'acroissement de la population par consommation de proies, et γ le taux le mortalité naturelle. Ce modèle proie-prédateur a fait l'objet de nombreuses études formelles. Cependant la recherche analytique de solutions montre rapidement ses limites si l'on introduit une troisième population (ou davantage) dans le système.

La disponibilité d'ordinateurs de plus en plus rapides depuis les années 50, a ouvert la voie au développement de méthodes de modélisation basées sur la simulation. Lorsque la résolution des équations s'avère trop difficile (voire impossible) de manière analytique, on peut faire appel à des techniques d'intégration numérique. Ces techniques (les méthodes d'Euler ou Runge-Kutta sont les plus connues) permettent, par une discrétisation des calculs, d'obtenir des solutions approchées. On choisit comme condition initiale une valeur pour chaque variable, puis on reproduit les changements d'état de ces variables en calculant leurs valeurs par itérations successives : il s'agit d'une simulation. L'algorithme qui décrit la façon de calculer les valeurs successives est le *modèle de simulation* qui représente le modèle analytique et donne accès à une approximation des solutions de celui-ci.

Ces techniques d'intégration numérique ont ouvert la voie au développement de logiciels de modélisation de systèmes dynamiques et de simulation de plus en plus évolués. Il faut noter cependant que cette évolution a emprunté deux voies différentes, qui correspondent chacune à un point de vue sur la façon de se représenter un système pour en construire un modèle :

L'une consiste à raisonner avec une vision différentielle des choses. On pense en terme de variables continues qui représentent par exemple des grandeurs physiques comme une température, ou une intensité électrique, et on exprime des contraintes ou des liens entre ces variables à l'aide d'équations différentielles. C'est que l'on a coutume d'appeler simplement la modélisation de systèmes dynamiques.

L'autre consiste à raisonner avec une vision intégrative des choses. On pense en termes de variables agrégées, qui représentent des quantités comme un niveau d'eau ou le nombre d'individus dans une population, et on exprime sous forme d'équations les flux de matière, d'individus, ou d'énergie qui circulent d'une variable agrégée à l'autre. Cette façon de voir les choses a été proposée par Forrester dans les années soixante et il l'a nommée la *Dynamique des Systèmes*[3]. Nous allons détailler davantage cette approche parce qu'elle a été beaucoup utilisée dans les disciplines qui constituent le contexte thématique de nos travaux.

Forrester explique sa préférence pour une vision intégrative de la façon suivante [57] : "*Formuler un système en termes d'équations différentielles rend confus, pour de nombreux étudiants, la direction de causalité dans les systèmes [...] Prenons par exemple les relations entre position, vitesse et accélération. Voir la vitesse comme la pente, ou dérivée, de la position par rapport au temps peut suggérer que le changement de position est responsable de la vitesse au lieu de l'inverse. La direction de causalité apparait plus clairement lorsque la description du système commence par la force qui cause l'accélération, puis que l'on intégre l'accélération pour produire la vitesse, et l'on intégre la vitesse pour produire la position.*"

Ce souci de clarté dans la compréhension et la modélisation des systèmes l'a amené à concevoir une méthode de modélisation basée sur un formalisme graphique. Elle permet de représenter les notions de *stock*, qui intègre l'évolution temporelle d'une variable, et de *flux*, c'est à dire les échanges de matière ou d'énergie entre des stocks. Les circuits de circulation d'informations sont aussi pris en compte dans ce formalisme, de façon à exprimer l'influence de l'information sur des fonctions de décision. Le fait de savoir qu'une variable atteint un certain seuil par exemple peut influencer une fonction de décision qui va agir sur le débit d'un flux dans le système. Forrester cherchait en particulier à ce que l'on puisse construire un modèle en conservant une vision d'ensemble des boucles de rétroactions d'un système.

Les équations correspondantes pouvaient ensuite être déduites du formalisme gra-

[3]Le risque de confusion entre modélisation de systèmes dynamiques et Dynamique des Systèmes est amplifié par le fait que ces termes sont très souvent utilisés en anglais où ils se retrouvent inversés : dynamic systems (ou DS) pour le premier cas et System Dynamics (ou SD) pour le second.

phique, et un outil de résolution numérique (nommé Dynamo) était proposé pour effectuer des simulations à partir de ces équations.

Plusieurs outils de modélisation et simulation basés sur les principes de la Dynamique des Systèmes ont depuis été développés, tels que Stella [37] ou plus récemment Simile[109]. La plupart de ces outils ont en commun un environnement graphique permettant la construction de modèles de type stock-flux, leur traduction automatique sous une forme exécutable, un moteur de calcul des simulations, et diverses formes de restitution des résultats.

La Dynamique des Systèmes a été d'abord appliquée par Forrester lui même pour mener une réflexion sur la gestion de l'urbanisme[58] et pour une étude de l'évolution des sociétés humaines à l'échelle mondiale intégrant de grand mécanismes sociaux, économiques, techniques, et écologiques [59].

Ces premières applications illustrent les caractéristiques principales de la modélisation en Dynamique des Systèmes : les variables que l'on manipule représentent essentiellement des agrégats et l'on ne distingue donc pas d'individualité. On doit réfléchir aux dépendances structurelles globales du système et fournir des données quantitavives en conséquence.[25]

Avec la modélisation de systèmes dynamiques selon une vision différentielle des choses : on manipule des grandeurs physiques dont l'état évolue de manière continue. On n'a donc pas cette notion d'agrégat ni non plus celle d'individualité.

Dans les deux cas on construit un modèle essayant d'abord de se représenter une vision globale du système que l'on étudie. On essaye d'en distinguer les mécanismes généraux, puis éventuellement de détailler chacun d'eux en sous-mécanismes, et ainsi de suite. Ainsi, la modélisation de systèmes dynamiques et la Dynamique des Systèmes sont des approches souvent caractérisées de *descendantes*. On notera aussi que dans la plupart des cas dans les approches descendantes, la structure du modèle, c'est à dire l'ensemble des liens que l'on introduit entre les variables, est statique.

Le fait de ne travailler que sur des variables agrégées est apparu comme une contrainte dans certaines situations de modélisation, notamment lorsque l'on désire tenir compte de spécificités locales des comportements d'un système. Ce besoin d'introduire des individus dans les modèles a amené une évolution des outils de modélisation basés sur des approches descendantes.

Avec SME (Spatial Modelling Environment) [99, 38] par exemple, il est possible de modéliser un ensemble d'individus qui sont les cellules d'une grille. Le comportement de chaque individu est un modèle de type stock-flux et certains flux peuvent donc être horizontaux, c'est à dire passer d'une cellule à une de ses voisines.

SIMILE [109] est un autre exemple d'évolution vers l'usage de variables moins agrégées. Cet outil propose lui aussi la possibilité de gérer des individus et de décrire leur comportement par des modèles de type stock-flux, mais il propose en outre un certain nombre de fonctions de désagrégation. Ces dernières permettent de transformer une variable de stock en une décomposition de sous modèles et l'on peut éventuellement donner

une position dans une grille aux éléments de ces sous-modèles.

2.3 Modélisations individu centrées

Aux approches descendantes que nous venons de voir, on peut opposer une conception *ascendante* de la modélisation. On décrit dans ce cas les règles de comportements d'entités élémentaires d'un système (individu, cellule, agent, ...) et l'on simule les interactions d'un grand nombre de ces entités pour observer l'évolution de l'ensemble du système.

Les deux principales formes de modélisation de ce type sont les automates cellulaires, et les systèmes multi-agents.

2.3.1 Automates cellulaires

Imaginés dans les années 1960 par J. von Neumann et S. M. Ulam [116] les automates cellulaires font partie des rares paradigmes de simulation qui intègrent dans leur définition même, une prise en compte de l'espace.

Un automate cellulaire se définit d'une manière générale par :

- Un réseau de cellules dans un espace de dimension D.
- Un ensemble d'états possibles pour ces cellules.
- Une ou plusieurs règles de transition locales, qui décrivent comment calculer l'état d'une cellule à un instant t à partir de l'état de cette même cellule et de l'état des cellules voisines à l'instant $t-1$. La définition du voisinage pouvant être différente selon les besoins, comme par exemple les voisinages de von Neumann et celui de Moore qui sont les plus couramment utilisés (figure 2.1)

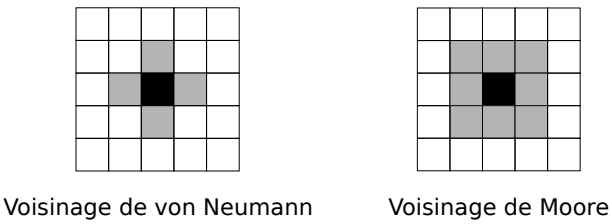


FIG. 2.1 – Deux formes de voisinage les plus couramment utilisées dans les automates cellulaires

La transition d'état, entre les temps t et $t+1$, de l'ensemble des cellules est supposée simultanée sur tout le réseau. Les logiciels de construction d'automates cellulaires permettent pour la plupart une expression déclarative des règles de transition : la programmation de l'automate se présente sous la forme d'une liste de règles de transition, sans ordre particulier. Le modélisateur n'a pas besoin de programmer le parcours des cellules du réseau pour calculer les changements d'états comme on le ferait avec un langage impératif séquentiel. Cela suppose que le moteur d'exécution des règles de transition prenne en charge l'application des règles à l'ensemble des cellules, et qu'il garantisse un

résultat identique quel que soit l'ordre dans lequel chaque cellule est visitée. Nous verrons d'autres langages déclaratifs à propos des systèmes de ré-écriture dans la section 2.4.1.

L'automate proposé à l'origine par von Neumann comportait 29 états. Il n'a pas été implémenté mais a servi à démontrer qu'il était possible avec ce type d'outil de programmer n'importe quelle instruction et donc de simuler (théoriquement) le fonctionnement d'un ordinateur. Par la suite de très nombreux automates cellulaires plus simples ont été imaginés, implémentés et étudiés.

Le modèle *Life* (connu en français sous le nom de *Jeu de la Vie*) proposé par J.H.Conway en 1970 [66], a en particulier suscité un grand intérêt auprès de la communauté scientifique.

Dans le Jeu de la Vie, le réseau est une grille de dimension 2 à cellules carrées. Il n'y a que deux états possibles : une cellule est soit vivante, soit morte. On a deux règles de transition locale qui s'appliquent sur un voisinage de Moore (huit cellules) :

- Une cellule passe à l'état vivant si elle est entourée de trois cellules vivantes.
- Une cellule passe à l'état mort si elle est entourée de moins de deux ou plus de trois cellules vivantes.

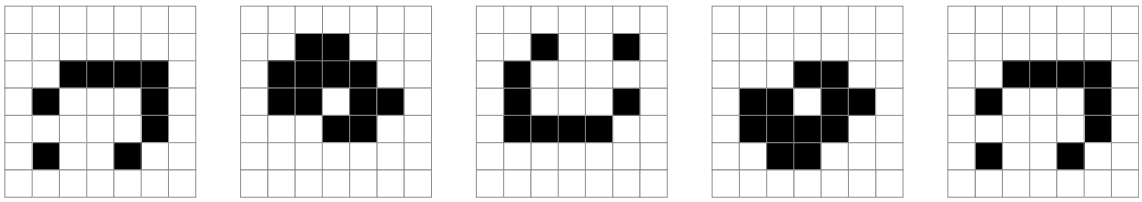


FIG. 2.2 – Exemple de motif répétitif dans le Jeu de la Vie

Ces deux règles de transition suffisent à produire des effets qu'il aurait été difficile d'imaginer sans faire appel à la simulation. C'est par exemple la production de certains motifs qui se répètent ou se déplacent sur la grille : il s'agit d'un phénomène observable sur des groupes de cellules, dans une configuration particulière d'états (figure 2.2). L'émergence de ces comportements de groupes de cellules n'est pas seulement due aux règles individuelles, mais aussi à la configuration de voisinage d'un certain nombre de cellules les unes par rapport aux autres. Ce phénomène d'émergence est une des caractéristiques des systèmes complexes. La facilité avec laquelle on peut construire un simulateur spatialisé de système complexe, et l'aspect graphique des résultats sont des facteurs qui font l'attrait des automates cellulaires.

Pour illustrer les effets globaux de règles définies localement, on peut prendre en exemple le modèle de ségrégation de T. Schelling [133] qui peut être aisément implémenté sous la forme d'un automate cellulaire. Ce modèle décrit l'évolution de la distribution spatiale de deux populations distinctes (blanc ou noir, garçons ou filles, officiers ou hommes du rang, ...) à partir de règles de préférence locale de voisinage. Chaque individu peut se déplacer de façon a essayer d'avoir une certaine proportion de proches voisins qui sont de la même catégorie que lui. Les simulations montrent que même si les individus sont très

tolérants vis à vis de leur voisinage, c'est à dire qu'ils préfèrent être entre voisins d'une même catégorie mais en très légère majorité par rapport à l'autre catégorie, on aboutit à une situation de ségrégation importante à l'échelle globale au bout d'un certain temps. Pour des individus tolérants, cette ségrégation n'était pourtant pas recherchée. Le modèle montre aussi que l'on aboutit à une importante ségrégation dans les cas ou les individus acceptent d'être mélangés en propotion égale avec l'autre population, mais que l'une des populations est légèrement plus nombreuse [132].

La simplicité de mise en oeuvre des automates cellulaires s'accompagne malheureusement de contraintes importantes. Les automates cellulaire sont des modèles dynamiques à structure statique : la topologie du réseau de cellules ne change pas durant une simulation et les règles de voisinage non plus. La taille des cellules est fixe et rend difficile la modélisation de phénomènes impliquant des interactions à différents niveaux d'échelles. Les règles de transition sont identiques pour toutes les cellules.

Nous allons voir que certaines de ces contraintes peuvent être levées avec les systèmes multi-agents.

2.3.2 Systèmes multi-agents

Les systèmes multi-agents (SMA) sont issus des travaux relatifs aux systèmes distribués en intelligence artificielle. C'est d'abord le concept d'*acteur* qui fut introduit par C. Hewitt en 1976 [83] : un acteur est une entité qui a la capacité de communiquer avec d'autres acteurs à travers l'échange de messages. Un acteur est défini par les actions qu'il peut effectuer lors de la réception d'un message, et par l'ensemble des liens vers d'autres acteurs avec lesquels il est en mesure de communiquer. L'objectif était essentiellement d'étudier les méthodes de partage de connaissances et les échanges d'information entre groupes d'acteurs. De tels systèmes sont entièrement définis par les types de messages que les acteurs peuvent échanger, les liens entre acteurs, et les actions prises par les acteurs lors de la réception de messages. La notion d'environnement n'était pas présente dans ce type de système.

Le concept d'*agent* et celui de *système multi-agents* ont ensuite été proposés par L. Gasser et al. en 1987 [69], avec un outil nommé MACE (Multi-Agent Computing Environment), là encore pour servir les besoins de travaux en intelligence articielle distribuée.

Comme les acteurs, les agents sont capables de communiquer entre eux, mais ils ne se contentent pas de réagir aux messages qu'ils reçoivent : ils ont des objectifs propres qu'ils cherchent à atteindre et ont pour cela une certaine autonomie dans leur capacité de décision. Les agents évoluent dans un environnement qui peut servir d'espace pour se déplacer mais aussi de lieu dans lequel il est possible de puiser des ressources (nourriture, énergie, information localisée). Pour interagir avec cet environnement les agents sont dotés de moyens de perception (des capteurs), et de moyens d'action sur l'environnement (des effecteurs). Les informations collectées par ces moyens de perception sont utilisées par les agents pour se constituer une représentation du monde dans lequel ils évoluent. Cette représentation qu'ils possèdent en interne, et les messages qu'ils recoivent d'autres agents avec lesquels ils communiquent, servent d'abord à estimer s'ils se rapprochent de leurs

objectifs ou s'ils s'en éloignent, et ensuite à décider des actions à effectuer pour réduire l'écart entre la situation actuelle et les objectifs à atteindre.

Il faut noter que le concept d'agent est plus général que celui d'individu. Un agent peut être un individu mais peut aussi représenter une entité à un autre niveau d'organisation (une cohorte, un village, etc.) [27]. Les systèmes multi-agents ne se contentent donc pas d'une modélisation de comportements individuels, mais couvrent aussi les processus de communication, de coordination, et d'organisation sociale entre agents.

Les éléments de description des systèmes multi-agents que nous venons de donner correspondent à ce que l'on attend de tels outils en général. Le lecteur intéressé par un aperçu général plus complet pourra se référer à l'ouvrage de J. Ferber [52]. Dans le détail, les définitions sont nombreuses, et cette variabilité est largement imputable à la grande diversité de domaines d'application qui se sont emparés de ce paradigme de modélisation [60].

Ce succès a apporté une abondance de modèles et a donné de la matière et le recul nécessaire pour effectuer des analyses critiques de la façon de concevoir et d'utiliser les SMA [77, 44]. Ces travaux ont abouti par la suite à la mise au point du protocole ODD (Overview, Design concepts, and Details) dédié à la description de modèles individu centrés et de SMA [78, 79]. Il faut en effet observer que si les formes de modélisations descendantes peuvent en général s'exprimer à l'aide de formalismes mathématiques, les formes ascendantes sont constituées de règles de comportement qui sont plutôt exprimées sous la forme d'algorithmes et de leur implémentation dans un langage de programmation. Cette absence d'un formalisme unique rend plus difficile la compréhension par des tiers, leur capacité à reproduire un modèle ou à le réutiliser, leur capacité à comparer des modèles. L'objectif du protocole ODD est de fournir un cadre pour la description des modèles individu centrés, qui à défaut d'être complètement formel, favorise les échanges entre chercheurs avec une forme de présentation qui soit indépendante des outils informatiques utilisés[4].

Les méthodes et outils d'aide à la conception de SMA, à leur construction, leur implementation et à la simulation font eux aussi l'objet de travaux de recherche. Un des domaines les plus actifs concerne la représentation des connaissances qui se décline à travers différents aspects de méta-modélisation :

- La définition et la spécification de l'identité des agents eux mêmes, c'est à dire les concepts qu'ils représentent, ce qu'ils sont et ce qu'ils ne sont pas.
- La représentation que possèdent les agents cognitifs du monde qui les entoure. Ils font appel à cette représentation du monde pour prendre des décisions et agir pour tenter d'atteindre leurs objectifs. Cette représentation peut être plus ou moins complexe, plus ou moins abstraite.
- Les rôles sociaux (voir le paragraphe *Organisation et concept de rôle* ci-dessous), obligations et droits des agents peuvent eux aussi être specifiés à l'aide de méta

[4]Le protocole ODD a été conçu par une communauté de modélisateurs en écologie et sciences environnementales. L'industrie fait aussi usage de systèmes multi-agents et a produit des spécifications à un niveau plus technique, à travers un comité de normalisation (nommé FIPA) qui fait partie de l'IEEE [85]

modèles, d'ontologies, de règles d'inférences.
La plateforme Mimosa [114] est un exemple de réalisation qui tente d'établir le lien entre représentation des connaissances et modèle de simulation à base d'agents.

Il n'est pas nécessaire ici de traiter toutes les facettes des recherches sur les SMA, mais nous avons choisi d'en détailler deux aspects qui présentent un intérêt particulier pour nos travaux :

1. la constitution de niveaux organisationnels entre groupes d'agents et le concept de rôle
2. la modélisation d'un environnement dans lequel des agents évoluent et les liens entre les agents et cet environnement

Organisation et concept de rôle

Comme avec les automates cellulaires, les systèmes multi-agents permettent de définir des règles de comportements individuels puis d'observer l'émergence de comportements de groupes. Ce phénomène d'émergence est recherché et étudié dans certaines applications, en particulier pour l'étude de systèmes complexes pour lesquels la simulation est la seule méthode disponible qui donne accès à l'observation de ces phénomènes. Dans d'autres applications, le caractère imprévisible des comportements de groupe pouvant émerger de modèles basés uniquement sur des règles de comportements individuels est davantage considéré comme un problème. C'est par exemple le cas lorsque l'on se base sur les principes des SMA pour construire des architectures de systèmes distribués [87]. On cherche alors à régler ce problème en ajoutant des formes d'organisations contrôlées de groupes d'agents, des méthodes de coordination de leurs actions, et des moyens de définir des objectifs communs.

Dès les premiers travaux sur les SMA, Gasser et al. [69] ont d'ailleurs proposé une forme d'organisation permettant de coordonner les actions de groupes d'agents. D'autres formes ont été imaginées depuis et L. Gasser a publié en 2001 une analyse critique des travaux de recherche sur le concept d'organisation dans les SMA [68]. Il distingue trois façons d'aborder cette question : technologique, phénoménologique, et théorique qui ne sont d'ailleurs pas mutuellement exclusives.

- Le point de vue technologique est relatif aux économies d'échelles qu'une organisation permet d'obtenir. C'est aussi l'idée que l'union fait la force : en se regroupant on peut atteindre des objectifs inaccessibles à des individus isolés comme accéder à des ressources sur une large portion d'espace, déplacer de grandes quantités de matière ou mener à bien des projets dont la durée dépasse la durée de vie d'un individu.
- Le point de vue phénoménologique nous met dans la posture d'un observateur de phénomènes émergents : on voit des organisations se constituer, exister et évoluer. Ces organisations sont dans ce cas un objet d'étude.
- Le point de vue théorique se situe au niveau de la conception des SMA : on cherche à définir des concepts abstraits correspondant aux différentes formes d'organisation,

leurs propriétés, les processus dont elles dépendent. Cela permet la mise au point de modèles conceptuels dont l'objectif est de simplifier (et parfois automatiser) la construction de modèles à base de SMA.

Les travaux qui ont abordé la question des organisations d'agents sous l'angle théorique ont en commun des concepts structurels, qui permettent de spécifier la nature d'une organisation d'agents (une collection, un groupe, une société par exemple) et le rapport des agents avec une organisation (leur rôle, d'éventuels liens entre eux). Coutinho et al. [40] ont montré que dans certains modèles conceptuels, ces éléments structurels sont complétés par d'autres éléments d'ordre fonctionnels (les objectifs d'une organisation), normatifs (les droits, les obligations), et dialogiques (les protocoles et régles de communications). A titre d'exemple nous présentons ci-dessous les éléments des modèles conceptuels Gaia, MOISE+ et AGR :

Dans Gaia [152], un système est constitué de rôles et d'interactions. Le concept de rôle est défini précisément comme un ensemble de responsabilités, de permissions, d'activités et de protocoles. Il représente en fait une fonction abstraite dont le sens est comparable à celui d'une profession dans une entreprise. L'ensemble des responsabilités associées à un rôle détermine ses objectifs. Un agent est une entité concrète qui peut exercer un ou plusieurs rôles. La notion d'interaction étant ici définie comme une relation abstraite entre rôles (des liens de dépendance fonctionnelle par exemple) et non entre agents.

Dans MOISE+ [84] on trouve des éléments structurels (rôle, relation, groupe), des éléments fonctionnels (objectifs, missions) et des éléments *déontiques* qui correspondent à un aspect normatif (permissions et obligations). Le rôle est ici défini comme un ensemble de contraintes qui s'appliquent à une entité jouant ce rôle : des contraintes de relation vis à vis d'autres entités d'un groupe, et des contraintes déontiques relatives aux objectifs généraux du groupe. Ces dernières contraintes sont exprimées sous la forme de missions distribuées aux agents.

Dans AGR [81, 53] on ne trouve que des éléments de structure. Il est basé sur les concepts d'Agent, Groupe et Rôle. Les niveaux organisationnels sont des groupes. Un groupe est défini comme une communauté d'agents. Un rôle est la représentation abstraite d'une fonction du groupe pouvant contraindre le comportement de l'agent. Un agent peut apparaitre dans différents groupes et jouer un rôle différent dans chacun de ces groupes. Le modèle AGR a volontairement été conçu de manière minimaliste, il a été complété par la distinction de niveaux descriptif (des comportements et interactions pouvant intervenir dans le système) et exécutif (chargé de la mise en oeuvre de ces comportements et interactions) dans le modèle MOCA [11].

Liens entre Agents et leur environnement

L'environnement est présent dans toutes les définitions du concept d'agent, en particulier parce que les agents ont une capacité à percevoir leur environnement, et à agir sur celui-ci. Si l'on s'en tient à cela, tout ce qui est hors de l'agent et que celui-ci peut percevoir fait partie de son environnement. C'est semble-t-il la vision de Gasser et al. [69] pour

qui un agent existe dans un environnement, et celui-ci est constitué du reste du système, des autres agents, et même du reste du monde autour du système. Il n'y a donc aucune contrainte par définition sur la forme que peut prendre l'environnement des agents dans un SMA. Les modélisateurs sont libres de construire l'environnement dont ils ont besoin, mais dans les faits très peu de travaux ont été consacrés à l'étude de ces environnements [52, 96].

Dans la pratique, on constate que l'environnement prend souvent la forme d'un espace dans lequel des agents mobiles peuvent se déplacer. J. Ferber [52] indique par exemple qu'en général l'environnement est un espace disposant d'une métrique. Il ajoute que celui-ci peut être soit central, soit distribué. Dans le premier cas l'environnement est modélisé à l'aide d'une seule entité à laquelle tous les agents ont accès, dans le second cas, l'environnement est modélisé comme un assemblage de plusieurs entités et les agents n'ont accès qu'à un sous-ensemble de ces entités qui constitue un voisinage.

La modélisation de cet espace peut être indépendante du SMA. C'est le cas du "Sugarscape" imaginé par Epstein et al. [49] qui propose de coupler des agents à un automate cellulaire. Ceux-ci peuvent se déplacer sur la grille de l'automate cellulaire et consommer des ressources présentes dans les cellules.

La modélisation de l'espace peut aussi être intégrée au SMA, en faisant en sorte que les éléments de cet espace soient eux même des agents. C'est la solution mise en oeuvre dans la plateforme de modélisation NetLogo[150] : l'espace est une grille dont chaque cellule est un agent immobile. Des agents mobiles peuvent se déplacer sur cette grille et communiquer avec les autres agents présents sur la même cellule, avec l'agent immobile qui incarne la cellule, ou avec ceux des cellules voisines.

Nous avons constaté en fait que la forme choisie pour modéliser l'environnement des agents est souvent induite par ce que propose la plateforme ou l'outil de modélisation utilisé. Pour avoir un aperçu de l'étendue de ces choix, nous avons examiné les solutions proposées par un certain nombre d'outils de modélisation multi-agents : Ascape/Escape [121], Cormas [26, 94], MadKit [82], MASON [95], Mass [86], Mimosa [115], NetLogo [150], RePast Symphony [117], Swarm [103]

Ce travail nous a permis de faire les constats suivants :

- Dans la quasi totalité de ces outils, la forme d'environnement privilégiée, c'est à dire souvent proposée par défaut et qu'il est facile de mettre en oeuvre, est de type spatial discret : une grille de cellules carrées en 2 dimensions. Certains outils donnent aussi les moyens de construire des grilles en 3 dimensions.
- Il est souvent possible de construire des graphes de liens directs entre agents. Cela donne accès à la construction de topologies de relations et de voisinages très variés, mais le modélisateur doit programmer lui même la construction du graphe et son éventuelle évolution.
- L'usage de données géographiques issues de systèmes d'information géographiques est rendu possible dans les deux tiers des outils, parfois sous la forme d'une extension. La mise en oeuvre d'environnements spatiaux de ce type demande souvent un investissement important de la part du modélisateur. Les opérateurs spatiaux que l'on trouve dans les systèmes d'information géographiques sont au pire absents (auquel cas l'environnement est statique et ne sert que de support aux agents), et au

mieux disponibles par programmation en accédant directement à des bibliothèques de fonctions dans un langage de programmation généraliste (comme Java ou C++).

2.4 Autres paradigmes et formalismes de modélisation

2.4.1 Systèmes de réécriture et MGS

Les systèmes de réécriture

Les systèmes de réécriture (SR) sont des outils abondamment utilisés en informatique, en particulier pour manipuler des expressions et des langages, les analyser, les transformer. Nous allons voir un usage original des SR permettant de modéliser des systèmes dynamiques à structure dynamique à l'aide d'un langage déclaratif.

Le principe général d'un SR consiste à repérer des motifs dans un ensemble structuré et à les remplacer par d'autres éléments dans cet ensemble. La spécification des motifs à rechercher et de ce qu'il faut écrire à leur place est donné sous la forme de règles que l'on appelle des règles de réécriture.

Ces règles ont la forme suivante : $l \rightarrow r$
En partie gauche de la règle, l décrit le motif à rechercher et qui doit être remplacé par la partie droite : r.

Si l'on se donne un alphabet, par exemple $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$, une règle peut exprimer directement la réécriture de mots constitués de cet alphabet.
Par exemple la règle $00 \rightarrow 0$ appliquée sur le mot 003 produira le mot 03.

On fait plus souvent appel à des règles qui comportent une ou plusieurs variables. Ces variables représentent n'importe quel *terme* (sous ensemble structuré) de la structure de départ.
Par exemple la règle $0x \rightarrow x$ appliquée sur le terme 03 produira 3.

Si l'on choisit d'appliquer la même règle sur le terme $t = 0040002$ on obtiendra le terme $t_1 = 04002$. En répétant cette action sur les termes successifs produits, on obtient $t_2 = 402$ et enfin $t_3 = 42$. Il n'est pas possible d'aller plus loin puisque l'on n'a plus de terme correspondant à la partie gauche de la règle.
Chaque application successive de la règle est une *dérivation* et se note $t_n \rightarrow t_{n+1}$. Pour ce SR, tous les t_i sont considérés comme équivalents, ils dérivent finalement tous vers 42 qui correspond à une *forme normale*. On dit que t est une forme normale si il n'existe pas de t_i tel que $t \rightarrow t_i$.

Il n'est pas toujours possible d'arriver à une forme normale. Les dérivations peuvent parfois être infinies, ou tourner en rond indéfiniment dans une suite répétitive de motifs.

Un SR possède la propriété de *normalisation* s'il existe une forme normale pour toute expression. Si de plus chaque expression possède une forme normale unique, le SR répond à la propriété de *confluence*.

Un système de réécriture est constitué de plusieurs règles, ce qui peut dans certains cas créer des situations de conflit. On doit alors ajouter de manière explicite des stratégies permettant de régler ces conflits, comme par exemple donner des priorités dans l'application des règles, ou placer des contraintes sur l'endroit où on les applique.

Lorsque l'on utilise un SR pour manipuler des expressions arithmétiques ou algébriques, on travaille avec des termes qui ont une structure arborescente. A chaque noeud de l'arbre, à commencer par sa racine, on fait correspondre un constructeur qui définit un certain nombre de fils. Chacun de ces fils est soit un noeud, soit une feuille de l'arbre.

Par exemple le terme $3 + 2 \times a$ peut correspondre à la construction de l'arbre : sum(3,mult(2,a)) (figure 2.3)

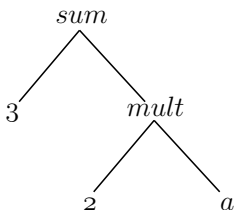


FIG. 2.3 – Représentation de l'arbre sum(3,mult(2,a)) correspondant au terme $3 + 2 \times a$

L'opération de filtrage, c'est à dire le parcours de l'arbre pour rechercher des motifs correspondants à la partie gauche des règles, s'effectue en partant de la racine et en associant les noeuds fils aux variables de la règle. Des appels récursifs au filtrage sont ensuite effectués sur ces variables si nécessaire, jusqu'à la consommation totale de l'arbre.

Le langage MGS

Un aspect des SR qui intéresse plus directement nos travaux est qu'ils peuvent être vus comme une façon de modéliser des systèmes dynamiques : l'état du système est représenté par une expression dont les sous-termes correspondent à l'état de sous systèmes, et des règles de réécriture décrivent des transformations locales qui permettent de spécifier l'évolution du système [136].

Par exemple la partie gauche d'une règle peut correspondre à un sous-système dont les éléments sont en interaction, et la partie droite de la règle correspond au résultat de cette interaction. Partant d'un état donné, les dérivations produites par les règles de transformation locale représentent une trajectoire possible du système dynamique.

Prenons l'exemple des cellules d'un organisme en développement : des échanges ont lieu entre des cellules voisines, à travers leurs membranes. Lorsque que les cellules se divisent, la topologie de la structure spatiale évolue : de nouveaux voisinages se forment créant de nouvelles possibilités d'échanges et d'interactions. En représentant ces opérations d'échange et de division cellulaire à l'aide de règles de réécriture, il est possible de modéliser l'évolution d'un tel système et en particulier l'aspect dynamique de sa structure.

Nous avons vu cependant que la conception des SR a été développée en priorité sur des termes ayant une structure arborescente. Représenter les structures topologiques d'un organisme en développement (pour rester sur cet exemple) sous une forme arborescente est loin d'être évident, et l'on peut être tenté d'imaginer d'autres formes de structures pour les termes représentant l'état d'un système [74].

C'est dans cet esprit qu'à été développé le langage de modélisation MGS [72]. Il s'agit d'un langage déclaratif, permettant de programmer des systèmes de réécriture, sur des structures de données qui sont des *collections topologiques*.

Une collection topologique est un ensemble d'entités, structuré par une relation de voisinage. Le voisinage définit ici à la fois les notions de localité et de sous-collection. Une sous-collection B d'une collection A est un sous-ensemble de A défini par un chemin (à travers les liens de voisinage) et qui hérite de l'organisation de A [73].

Des transformations locales peuvent être définies sous la forme de règles de réécriture qui spécifient les évolutions de sous-collections. L'application d'une règle du type $\beta \rightarrow f(\beta, \dots)$ à une collection A correspond aux actions suivantes :

1. selection d'une sous-collection B de A donc les éléments correspondent au chemin spécifié par β,
2. calcul d'une nouvelle collection C par application de la fonction f à B et ses voisins
3. insertion de C à la place de B dans la collection A.

Les travaux sur MGS ont été en partie formalisés dans la thèse de A. Spicher [135] : il s'est inspiré des concepts utilisés dans les outils de modélisation d'objets 3D, pour formaliser les collections topologiques sous la forme de complexes de chaînes.

2.4.2 Modélisation par événements discrets et le formalisme DEVS

Ziegler et al. [155, 154] ont abordé la modélisation de systèmes dynamiques de manière formelle en s'appuyant largement sur la théorie des systèmes. Cette formalisation considère d'abord trois aspects : une vision externe du système, sa structure interne, et la composition de plusieurs systèmes.

La vision externe peut s'établir sous la forme d'une observation historique de valeurs entrant dans un système et de valeurs qui en ressortent. La structure interne est caractérisée par des variables d'état et un moyen de spécifier les transitions d'états. La transition

d'état décrit comment les valeurs entrant dans le système pendant une certaine durée transforment l'état actuel en une succession de nouveaux états. Les valeurs sortant du système sont fonction de la succession de ces nouveaux états. Enfin la notion de composition est liée à la structure : un système peut être décomposé en plusieurs sous-systèmes connectés entre eux. Le couplage d'un certain nombre de sous-systèmes permet de constituer un système plus large. Une propriété importante de cette construction est la *fermeture par composition* : une même base théorique permet de spécifier les composants d'un système, et un système composé. Le système composé peut alors lui même faire office de composant dans un système plus large.

A cette base théorique, B.P. Ziegler a apporté une formalisation de la gestion du temps par événements discrets : ce sont les occurences d'événements discrets qui déterminent l'avancement du temps. Il a nommé ce formalisme : Discrete EVents systems Specification (DEVS).

Un modèle DEVS atomique est caractérisé par :

$$DEVS =< X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >$$

avec
- X est l'ensemble des ports et valeurs d'entrée
- Y est l'ensemble des ports et valeurs de sortie
- S est l'ensemble des états du système
- δ_{ext}, δ_{int} et δ_{con} sont les fonctions de transition externe, interne et de confluence
- λ est la fonction de sortie
- ta est la fonction d'avancement du temps

La fonction d'avancement du temps ta indique la durée pendant laquelle le modèle restera dans l'état S avant de passer à l'état suivant. Nous avons vu que le nouvel état est le produit d'une fonction de transition, et dans la version étendue de DEVS que nous donnons ici, la fonction de transition est divisée en trois fonctions :

δ_{int} La fonction de transition *interne* détermine l'évolution du modèle en l'absence de toute perturbation. Un modèle DEVS peut ainsi avoir une évolution propre, et changer d'état régulièrement même si aucun événement extérieur ne se produit. En absence d'interruption par un événement extérieur, la durée de transition de δ_{int} est ta.

δ_{ext} La fonction de transition *externe* est activée lorsqu'un événement intervient sur l'un des ports d'entrée du modèle DEVS. Cela signifie que lors de l'occurence d'un événement, si l'on note e le temps écoulé depuis la dernière transition, δ_{int} est interrompue, et c'est δ_{ext} qui prend le relais pour la durée $ta - e$ qui reste jusqu'à atteindre ta.

δ_{con} On fait appel à une fonction de transition *de confluence* [33] dans le cas ou un événement se produit exactement en même temps qu'un changement d'état ($e = 0$ ou $e = ta$) : on est en situation de conflit puisqu'on ne sait si le changement d'état doit être pris en charge par δ_{int} ou par δ_{ext}. Le modélisateur peut choisir de définir une fonction δ_{con} qui sera chargée de déterminer le nouvel état lorsque ce cas se présente.

Le formalisme DEVS permet d'exprimer des approches descendantes de modélisation. Il existe en effet plusieurs méthodes pour écrire un système d'équations différentielles avec DEVS. L'une consiste à faire appel à un formalisme unificateur DEVS&DESS[5] [154]. Une autre consiste à spécifier la simulation numérique d'équations différentielles à l'aide d'événements discrets [48].

Mais DEVS est suffisamment universel pour aussi permettre la description de formes ascendantes de modélisation. Ainsi la plateforme de modélisation multi-agents Mimosa développée au CIRAD [110, 115] est dotée d'une sémantique opérationnelle qui est basée sur le formalisme DEVS [111].

De nombreuses extensions au formalisme DEVS ont été proposées. Nous nous limiterons ici à citer deux exemples :

- Cell-DEVS [147, 148] qui apporte la capacité à formaliser les échanges d'événements discrets dans une grille de cellules,
- DS-DEVS [17] qui apporte les moyens de formaliser des modèles à événements discrets ayant une structure dynamique.

DEVS est donc un formalisme qu'il est possible d'adapter à des paradigmes de modélisation différents, ce qui en fait une base théorique pouvant servir au couplage de plusieurs paradigmes de modélisation (la multi-modélisation). R. Duboz a par exemple montré dans sa thèse [48] comment on peut formaliser avec DEVS un SMA à base d'agents réactifs, et réaliser le couplage avec des équations différentielles.

La plateforme VLE (Virtual Laboratory Environment) [127, 126] tire parti de cette qualité de DEVS pour fournir un environnement de multi-modélisation sous la forme de bibliothèques en C++. On peut d'ailleurs noter le développement récent d'extensions de VLE dédiées à la modélisation d'agro-systèmes, sous la forme de bibliothèques C++ complémentaires, qui constituent la plateforme RECORD [20].

Nous avons vu que DEVS est un formalisme qui laisse beaucoup de liberté aux modélisateurs quand à la gestion du temps dans les modèles, avec la possibilité de combiner temps discret et temps continu, tout en restant dans un cadre formel précis. En ce qui concerne la gestion de l'espace, la situation est semblable à ce que l'on a vu pour les SMA, c'est à dire que sont favorisés soit une représentation sous la forme de grille, soit un couplage faible avec un système d'information géographique. Une prise en compte plus complète de topologies spatiales et de leurs dynamiques reste possible en complément, à travers une programmation directe, mais cela sort en général du cadre proposé par ces outils.

[5] DESS : Discrete Event and Differential Equation Specification System

2.4.3 SIG et simulation

Sous le terme Systèmes d'Information Géographique (SIG) on regroupe traditionnellement un ensemble d'outils aux fonctionnalités complémentaires [75] :

- acquisition et saisie de données géoréférencées
- stockage et indexation de données géoréférencées
- représentation cartographique, extraction de données
- manipulation, et analyse d'information géographique. L'analyse étant l'usage principal dont les autres fonctions sont le support.

Fonctionnalités auxquelles peuvent parfois s'ajouter : la gestion de métadonnées, et des services web géographiques et fonctions d'interopérabilité.

Les SIG ont donc avant tout été conçus pour l'analyse spatiale et la représentation cartographique, mais les travaux de recherche liés à ces problématiques ont largement dépassé ces cadres. La prise en compte d'une composante temporelle en particulier, a fait l'objet d'une riche production scientifique ces vingt dernières années [92, 122, 153, 34, 146, 120]. Nous avons noté cependant que la plupart de ces travaux ont davantage abordé la question sous l'angle de la représentation des connaissances (avec par exemple le concept de *Pyramid framework* [101, 123]) et de la méta-modélisation [30, 120, 106].

La prise en compte du temps sous l'angle des processus, à des fins de simulation, n'est en revanche pas aussi développée. La plupart des logiciels récents de traitement de l'information géographique possèdent soit des APIs[6] dédiées, soit parfois un langage de programmation. On reste par ailleurs contraint par une représentation des structures spatiales héritée de la cartographie : l'information géographique est divisée en couches thématiques, chaque couche étant soit une grille d'une résolution donnée, soit en ensemble d'objets de type surface, ligne ou point. Les opérateurs disponibles sont surtout dédiés à l'analyse spatiale, la notion d'interaction n'est pas prise en compte dans ce contexte.

Il faut noter cependant qu'il existe des approches hybrides qui permettent de dépasser certaines des limites des SIG que nous venons d'indiquer. L'une des plus abouties est le langage SELES (Spatially Explicit Landscape Event Simulator) [50]. Il s'agit d'une forme de couplage entre gestion événementielle et SIG, qui se présente sous la forme d'un langage métier. L'information géographique est représentée sous la forme de couches thématiques qui sont des grilles de cellules. On est ici assez proche d'une modélisation ascendante, où il est possible de décrire si un événement va entrainer le changement d'état d'une cellule ou de groupes entiers de cellules. Cet outil est d'ailleurs recommandé par ses auteurs pour la création d'automates cellulaires, et de systèmes multi-agents dirigés par des événements.

[6] API : "Application Programming Interface", c'est à dire un certain nombre de définitions, de structures de données et de fonctions permettant de les manipuler, sous la forme d'une bibliothèque.

2.5 Conclusion

Nous avons vu que les approches descendantes de la modélisation nécessitent une compréhension de processus globaux qui régissent la dynamique d'un système. Ces processus sont alors décrits à travers des relations entre certaines variables agrégées de ce système, qui correspondent souvent à des grandeurs mesurables. La nécessité de pouvoir décrire des processus n'agissant que sur certaines parties de modèles, certaines catégories de variables, certains lieux spatialement délimités, voire même sur certains individus, a fait naitre des besoins en désagrégation de variables. Des méthodes de désagrégation ont été progressivement ajoutées aux outils basés sur des approches descendantes pour répondre à ces besoins.

Les approches ascendantes sont en revanche focalisées sur la description de règles de comportement portées par des individus. On met en oeuvre dans un modèle les interactions d'un grand nombre de ces individus, et la simulation permet d'observer certaines caractéristiques générales du système. Une évolution importante de ces approches a consisté à ajouter certaines formes d'organisations dans ces interactions, en décrivant par exemple des comportements de groupes d'individus. Diverses façons de former ces groupes ont été proposées qui ont été considérées sous des angles davantage sociaux ou fonctionnels que spatiaux. La composante spatiale, est essentiellement restée liée à la représentation d'un environnement dans lequel évoluent les individus. Cette composante spatiale s'est en outre avérée contraignante pour l'exercice de modélisation, en particulier lorsque que l'on doit faire à priori, le choix d'une forme de représentation spatiale.

Nous avons noté qu'il existe d'autres formalismes comme DEVS ou MGS qui peuvent être considérés comme complémentaires des formes ascendantes ou descendantes de modélisation, et seront certainement des sources d'inspiration pour la suite de nos travaux. Notre étude n'est pour autant pas exhaustive, et l'état de l'art n'est pas figé. Certaines propositions récentes méritent d'être suivies, parmi lesquelles nous voudrions citer deux exemples :

1. La modélisation considérée sous l'angle des processus telle que l'ont développée F. Reitsma et J. Albrecht [129], qui est une idée reprise récemment par W.S. Currie [42].
2. Une forme de modélisation dans laquelle les interactions sont basées sur des structures spatiales dynamiques et pouvant prendre différentes formes. Il s'agit des travaux de C. Gaucherel et al. qui ont été initiés sous la forme du langage L1 [70] et qui se poursuivent aujourd'hui à travers la plateforme de modélisation DYPAL en cours de construction.

Nous considérons nous aussi (comme les créateurs de MGS et DYPAL) que l'espace est structurant pour les interactions entre entités d'un modèle, mais que cette structure doit être prise en compte au même niveau que les organisations sociales, fonctionnelles ou hiérarchiques. Nous pensons nécessaire de pouvoir exprimer dans un modèle une compréhension de processus à plusieurs niveaux, selon différents points de vues, et en plusieurs

lieux. Nous estimons enfin nécessaire que ces structures soient dynamiques, et qu'il soit possible de les utiliser de manière sélective, c'est à dire que l'on puisse au cours du temps rendre effectives certaines interactions sur des sous groupes d'entités, pour par exemple rendre compte d'interactions locales, ou conjoncturelles.

Une telle souplesse dans la représentation des structures, ne peut être atteinte qu'en travaillant à un niveau très élémentaire, de façon à pouvoir intégrer les différentes formes d'interactions dans un formalisme commun.

Le concept de graphe nous est apparu suffisamment générique pour servir de brique de base à un tel formalisme. Nous proposons de tenter l'expérience de construire ces modèles sur des structures de graphes. Il nous faut proposer une façon d'intégrer à la fois la structure des interactions et leur nature dans un même outil. Cette approche de la modélisation n'est pas spécifiquement descendante ou ascendante, elle est plutôt conçue dès le départ à mi-chemin entre les deux.

Chapitre 3

Modélisation et graphes d'interaction

Sommaire

3.1 Introduction

Le contenu de ce chapitre répond au premier sous-objectif que nous nous sommes donné : proposer une forme générique de représentation des éléments et interactions d'un système. Nous y décrivons formellement les concepts de représentation que nous avons imaginés, qui s'appuient en particulier sur des graphes, et c'est la raison pour laquelle la première section de ce chapitre donne les éléments de théorie des graphes et le vocabulaire

associé dont nous faisons usage dans la suite de notre exposé.

A l'aide de graphes on peut naturellement modéliser les structures de différents aspects d'un système (c'est à dire qui peut interagir avec qui), comme ceux qui sont représentés sur la figure 3.1. Nous avons donc d'abord défini ce qui fait ces structures : de quoi sont constitués leurs éléments, comment leur état est représenté dans un modèle, et comment établir des liens entre ceux des éléments qui sont susceptibles d'interagir.

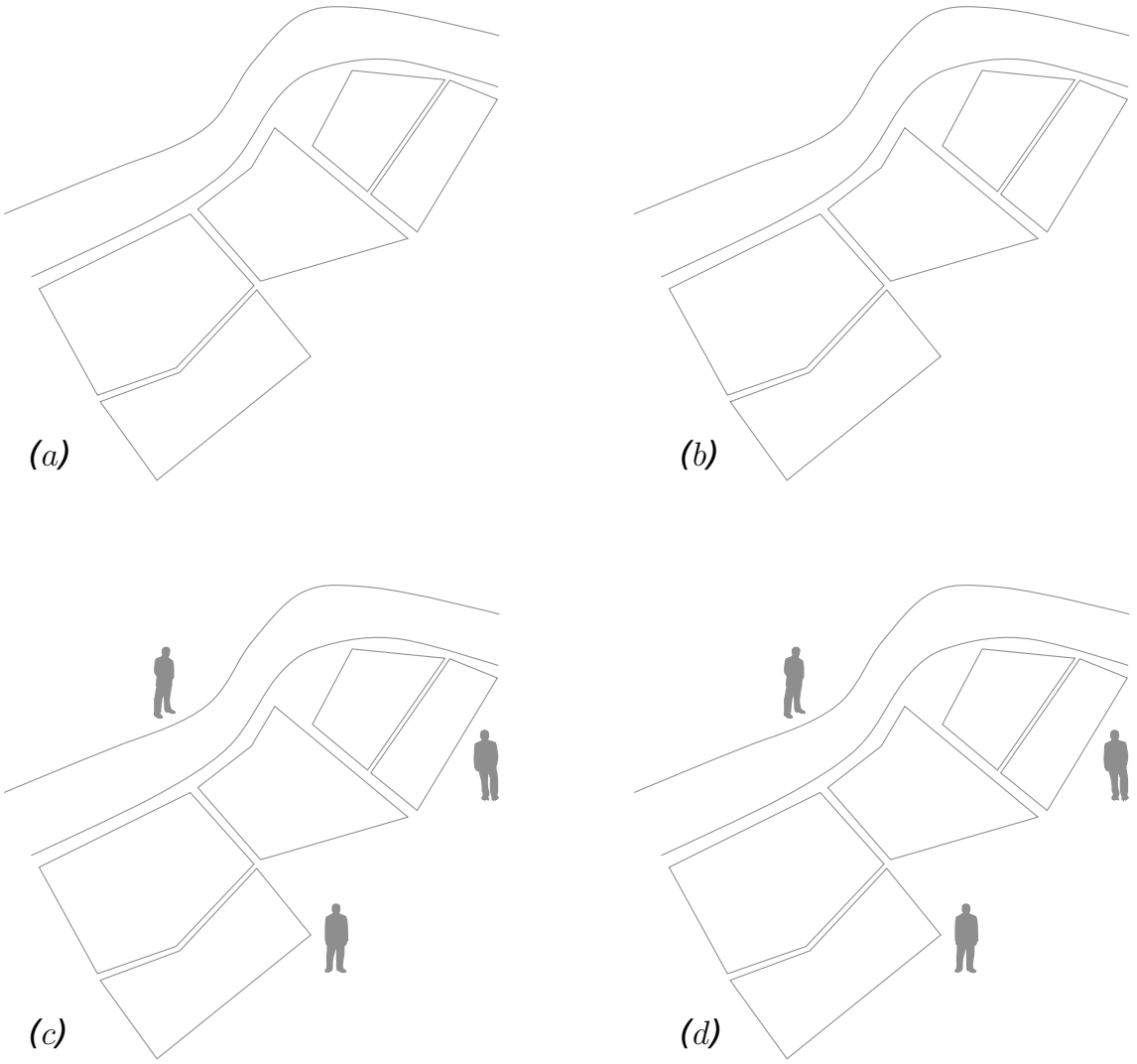


FIG. 3.1 – Différentes formes d'interactions représentées par des graphes : (a) et (b) sont des interactions basées sur un voisinage spatial entre des parcelles agricoles, et entre ces parcelles et une rivière. En (c) il s'agit d'interactions sociales entre acteurs d'un territoire, et en (d) des interactions fonctionnelles entre les acteurs et certains éléments du paysage sur lesquels ils peuvent agir. Les quatre graphes peuvent coexister dans un modèle pour représenter différents aspects des processus qui influencent la dynamique d'un paysage.

On peut remarquer que sur les schémas (a) et (c) de la figure 3.1, le liens sont repré-

sentés par de simples traits, alors que sur les schémas (b) et (d), nous avons utilisé des flèches. Cette différence reflète une différence de nature de leurs liaisons.

En (a) et (c) les éléments reliés sont de même type (des parcelles en (a), des acteurs en (c)) et l'on imagine qu'ils interagissent de manière symétrique : si une parcelle diffuse du pollen vers ses voisines par exemple, celles-ci peuvent aussi diffuser du pollen vers la première.

En (b) et (d) les éléments reliés sont de nature différente (rivière et parcelles en (b), acteurs et parcelles en (d)) et l'on imagine que leurs interactions sont asymétriques.

La nature des interactions est le deuxième aspect des concepts que nous avons imaginés. Cela constitue une partie de la sémantique d'un modèle : c'est la description de ce qui se passe lorsque des éléments interagissent les uns avec les autres et de la façon de spécifier cela précisément. Nous décrivons de manière formelle comment l'on peut faire porter cette sémantique par un graphe (que nous nommons alors *graphe d'interaction*), comment l'on peut distinguer les rôles joués par les éléments d'un modèle lors d'une interaction, et les différentes formes de fonctions qui peuvent constituer cette sémantique.

L'autre partie de la sémantique d'un modèle est liée à la dynamique d'un système. Nous précisons à quel niveau nous proposons de décrire l'enchaînement des opérations qui doit permettre de simuler l'évolution de ce système.

Les trois aspects que nous venons d'évoquer : la structure qui porte les interactions, leur nature et leur dynamique, font l'objet de la seconde section de ce chapitre.

Le fait de baser nos outils de modélisation sur des graphes, nous permet d'intégrer dans un même modèle des liaisons portant des interactions de toutes natures (spatiales, fonctionnelles, sociales, hiérarchiques). Cela comporte cependant le risque de rendre fastidieuse la spécification de modèles, justement parce que les graphes sont des constructions excessivement élémentaires. Nous verrons comment nous avons pris en compte ce risque dans la troisième section de ce chapitre. Nous y examinons les différentes questions relatives au passage de la donnée, dans ses diverses formes, aux graphes d'interactions que l'on souhaite manipuler dans un modèle.

3.2 Eléments de théorie des graphes

Nous donnons ici une définition formelle du concept de graphe, et de certaines propriétés et fonctions associées pour servir la compréhension de nos propos.

La théorie des graphes et sa formalisation font usage d'un vocabulaire riche. Nous ne présentons que les notions et notations qui nous sont utiles pour le contenu de ce document. Ces éléments sont principalement inspirés de [18, 24, 47, 107, 16], ouvrages auxquels le lecteur pourra se référer pour y trouver les nombreux autres aspects qui ne sont pas repris ici.

Le concept de *graphe* est intimement lié à sa représentation *graphique* (ce que reflète d'ailleurs la racine des mots) et c'est naturellement à une représentation graphique que l'on fait appel pour aborder ce concept de manière intuitive. Un graphe peut être vu comme un modèle de liens qui peuvent exister entre les éléments d'un ensemble. Ces éléments peuvent être représentés par des symboles (points, lettres, mots, dessins, etc.) et les liens sont représentés par des traits, des courbes, ou des flèches.

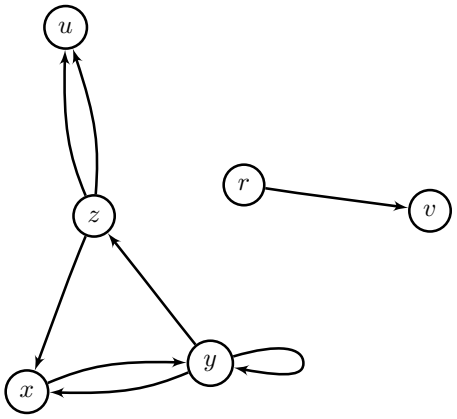


Fig. 3.2 – Exemple de graphe orienté

3.2.1 Définitions et vocabulaire relatifs aux graphes

On peut définir un *graphe* G de façon formelle par un couple : $G = (X, U)$ avec un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ et une famille $U = (u_1, u_2, \ldots, u_m)$ dont les éléments sont issus du produit cartésien : $X \times X = \{(x, y)/x \in X, y \in X\}$

Les éléments de X sont nommés les *sommets* [7] du graphe. Les éléments de U sont nommés les *arcs* du graphe.

Le nombre de sommets du graphe G est appelé l'*ordre* de G.

Il peut exister plusieurs arcs entre deux sommets, et si l'on admet un nombre maximum p d'arcs entre les sommets d'un graphe G, on dit que G est un *p-graphe*. Le graphe de la figure 3.2 par exemple est un 2-graphe d'ordre 6 qui comporte une boucle sur le sommet y.

Dans le cas des *1-graphes*, U est un ensemble (et pas seulement une famille) puisque les éléments u_i sont par définition distincts.

[7] Il semble qu'il n'y ait pas de convention générale sur l'usage du terme *sommet* et l'on trouve aussi le terme *nœud* dans la littérature. La situation n'est pas meilleure en anglais puisque les termes *node* et *vertex* sont indifféremment employés pour désigner les éléments de X. Le terme *nœud* semble cependant plus couramment employé dans le cas où un graphe représente un réseau, et nous avons donc choisi de retenir le terme *sommet* qui nous paraît davantage générique.

On dit qu'un graphe $G = (X, U)$ est *valué* lorsqu'à chaque arc est associé une valeur numérique, c'est à dire lorsqu'il existe une fonction de coût : $c : U \mapsto \mathbb{R}$.

Orientation

La distinction est souvent faite entre les concepts de *graphe orienté* et de *graphe non-orienté* . Selon Claude Berge [18] : "*tout graphe est orienté, mais pour des raisons conceptuelles, il est parfois peu commode de le considérer avec des lignes orientées si le problème posé est de nature non orienté.*".

Nous considèrerons que pour les usages non-orientés des graphes, on aura :

$$\forall (x, y) \in U, (x, y) \Leftrightarrow (y, x) \tag{3.1}$$

et nous utiliserons dans ce cas le terme *arête* pour désigner les arcs dont le sens n'a pas d'importance. Sur les figures, les arcs sont représentés par des flèches, et les arêtes par de simples traits.

Comme nous le verrons dans la section 3.3.3 de ce chapitre, nous allons principalement faire un usage orienté des graphes : un arc (x, y) ne sera en général pas équivalent à (y, x).

Arcs incidents, degré d'une extrémité

Pour un arc $u = (x, y)$, on appelle x *l'extrémité initiale* et y *l'extrémité terminale* de u. On dit que y est le *successeur* de x et que x est le *prédécesseur* de y. Les ensembles de successeurs et de prédécesseurs d'un sommet x sont notés respectivement :

$$\Gamma_G^+(x) \text{ et } \Gamma_G^-(x) \tag{3.2}$$

L'ensemble des sommets voisins d'un sommet x est noté :

$$\Gamma_G(x) = \Gamma_G^+(x) \cup \Gamma_G^-(x) \tag{3.3}$$

Un arc dont l'extrémité initiale et l'extrémité terminale sont identiques : $u = (x, x)$ est appelé une *boucle*.

Soit x un sommet, on appelle arcs *incidents* tous les arcs qui ont x pour extrémité. Il peut s'agir d'arcs incidents vers l'extérieur si x en est l'extrémité initiale, ou vers l'intérieur si x en est l'extrémité terminale.

Le nombre d'arcs incidents à une extrémité x est appelé *degré* de x et se note $d_G(x)$. Ce degré se décompose en $d_G^+(x)$, le *demi-degré extérieur* et $d_G^-(x)$, le *demi-degré intérieur*. On a donc : $d_G(x) = d_G^+(x) + d_G^-(x)$

Dans le cas d'une boucle (x, x), le même arc est à la fois incident à l'extérieur ($d_G^+(x) = 1$) et incident à l'intérieur ($d_G^-(x) = 1$). On a donc dans ce cas $d_G(x) = 2$.

Un graphe peut contenir des sommets qui ne sont reliés à aucun autre. Pour un tel sommet x on a alors $d_G(x) = 0$ et l'on dit que x est un sommet *isolé* du graphe.

Egalité et Isomorphisme

Soient les graphes $G = (X_G, U_G)$ et $H = (X_H, U_H)$, on dit que G et H sont identiques (noté $G = H$) si $X_G = X_H$ et $U_G = U_H$.

Il arrive que même si deux graphes ne sont pas identiques au sens strict, il soit possible d'en donner des représentations graphiques semblables. C'est le cas pour des graphes que l'on dit *isomorphes* :

Deux graphes $G = (X_G, U_G)$ et $H = (X_H, U_H)$ sont isomorphes (noté $G \simeq H$) si il existe une bijection $\varphi : X_G \to X_H$ telle que $\forall x, y \in X_G, (x, y) \in U_G \Leftrightarrow (\varphi(x), \varphi(y)) \in U_H$

Sous-graphes

Si l'on a un graphe $G = (X_G, U_G)$, on dit qu'un graphe $H = (X_H, U_H)$ est un *sous-graphe* de G lorsque $X_H \subseteq X_G$ et $U_H \subseteq U_G$. On le note $H \subseteq G$ et l'on peut dire que G *contient* H.

Un sous-graphe H de G est dit *couvrant* s'il contient tous les sommets de G, c'est à dire si l'on a $X_H = X_G$ et $U_H \subseteq U_G$.

Graphe complet et clique

Lorsque pour toute paire $\{x, y\}$ avec $x \in X, y \in X$ il existe au moins un arc de la forme (x, y) ou (y, x) on dit que le graphe est *complet*. Le sens des flèches importe peu dans cette définition, il s'agit d'une propriété non-orientée que l'on peut illustrer par un graphe comportant des arêtes comme sur l'exemple de la figure 3.3.

Un graphe complet de n sommets est noté K_n.

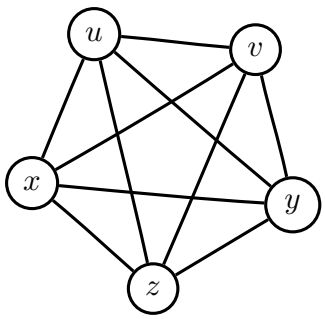


FIG. 3.3 – Graphe complet K_5

On nomme *clique* l'ensemble des sommets d'un sous-graphe complet. Un graphe K_n est aussi appelé une n-clique.

Chaines, chemins et cycles

On appelle *chaîne* une séquence d'arcs dans laquelle chaque arc possède une extrémité commune avec l'arc suivant dans la séquence. Dans une chaîne, il n'est donc pas tenu compte de l'orientation des arcs.

Une chaine qui ne passe pas deux fois par le même sommet, est appelée chaine *élémentaire*. Si elle ne contient pas deux fois le même arc, il s'agit d'une chaine *simple*. On appelle *cycle* une chaine simple dont les extrémités initiale et finale sont confondues.

Une chaine dont tous les arcs sont orientés dans le même sens est appelée *chemin*, et si cette chaine est aussi un cycle, on parle de *circuit*.

La *longueur* d'une chaine (ou d'un chemin) est égale au nombre d'arcs entre les deux extrémités de cette chaine.

Sur un graphe valué, le *coût* d'une chaîne (ou d'un chemin) est la somme des valeurs associées aux arêtes (ou aux arcs) de la chaine (ou du chemin).

Connexité

Un graphe non vide $G = (X, U)$ est dit *connexe* si $\forall x, y \in X$ il existe une chaine entre x et y.

Si un graphe G n'est pas connexe, les sous-graphes connexes de G sont les *composantes connexes* de G. L'exemple de la figure 3.2 comporte deux composantes connexes.

Si $\forall x, y \in X$ il existe un chemin entre x et y, on dit que G est *fortement connexe*.

Dans un graphe connexe, un sommet situé de telle façon que si on le retire, le graphe n'est plus connexe est un sommet *articulateur* ou *point d'articulation*. Un ensemble de sommets dont le retrait produit le même effet est un *ensemble d'articulation*. Une arête dont le retrait produit le même effet est appelée un *isthme*.

3.2.2 Quelques structures de graphes remarquables

Graphe r-parti

Lorsque pour un graphe $G = (X, U)$, on a une partition de X en r classes (avec $r \geqslant 2$), de façon à ce que chaque sommet d'un arc quelconque appartienne à une classe différente, on a un *graphe r-parti*.

Quand $r = 2$ on parle de graphe *biparti* (figure 3.4). Si X est constitué de deux classes X_1 et X_2 on peut noter le graphe bi-parti correspondant : $G = (X_1, X_2, U)$. Et le graphe bi-parti complet se notera $K_{n,m}$ avec n = card(X_1) et m=card(X_2).

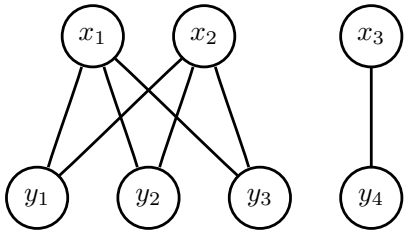


FIG. 3.4 – Graphe bi-parti entre les classes $\{x_1, x_2, x_3\}$ *et* $\{y_1, y_2, y_3, y_4\}$. Le sous-graphe constitué des classes $\{x_1, x_2\}$ *et* $\{y_1, y_2, y_3\}$ correspond au bi-parti complet $K_{2,3}$.

Un graphe r-parti dans lequel chaque sommet d'une classe est adjacent à tous les sommets de chacune des autres classes est un graphe r-parti *complet*.

Graphe planaire

Un graphe peut être représenté graphiquement sur un plan en associant chaque sommet du graphe à un point du plan et chaque arête à une courbe du plan qui relie les points correspondants à ses extrémités. Pour un graphe donné il existe une infinité de représentations graphiques possibles. On dit qu'un graphe est *planaire* lorsqu'il est possible d'en donner une représentation sur un plan de façon à ce que les courbes correspondant à ses arêtes ne se croisent pas.

La représentation d'un tel graphe sur un plan est appellé *graphe planaire topologique* et peut se définir de la façon suivante [47] :

Un graphe planaire topologique est un couple (X, U) d'ensembles finis vérifiant les propriétés :

i) $X \subseteq \mathbb{R}^2$;

ii) chaque arête relie deux sommets de X ;

iii) des arêtes distinctes ont des extrémités différentes ;

iv) une arête ne passe par aucun sommet ni aucun point d'une autre arête.

Un graphe planaire topologique (GPT) partitionne le plan en un certain nombre de régions adjacentes nommées *faces*. Une seule de ces régions est une face ouverte, qui correspond à l'extérieur du graphe. Les autres faces d'un GPT sont closes, entourées par des arêtes du graphe. Le graphe de la figure 3.5(a) est planaire, et l'on a une représentation possible de ce graphe sous la forme d'un graphe planaire topologique sur la figure 3.5(b).

Les graphes planaires topologiques possèdent certaines propriétés, dont la célèbre formule d'Euler[8] :

[8]Cette formule a été établie à l'origine pour les polyhèdres convexes. On peut trouver la démonstration de cette égalité pour les graphes planaires topologiques connexes dans [24, 47]

un graphe planaire topologique connexe possédant n sommets, m arêtes et l faces vérifie l'égalité

$$n - m + l = 2 \tag{3.4}$$

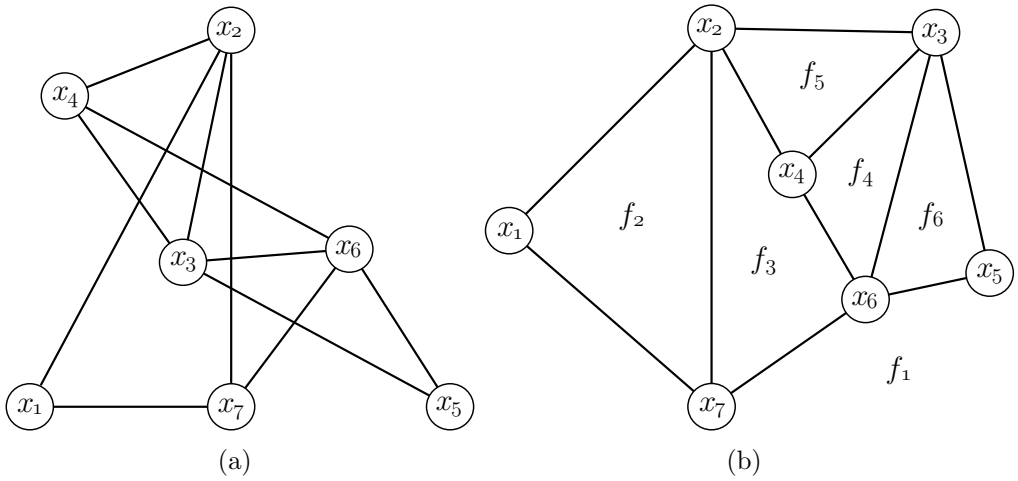


FIG. 3.5 – Graphe planaire (a) et sa représentation sous la forme d'un graphe planaire topologique comportant six faces : f_1 .. f_6 (b)

On peut déduire de cette formule que les graphes complets K_5 et $K_{3,3}$ ne peuvent être planaires. K. Kuratowski a proposé une caractérisation des graphes planaires qui fait usage de cette propriété. Il utilise la notion de subdivision d'un graphe : la subdivision d'un graphe G est un graphe G' obtenu en remplaçant certaines arêtes de G par des chemins contenant de nouveaux sommets.

Théorème de Kuratowski (dont [24] et [47] donnent une preuve) :
Un graphe G est planaire si et seulement si il ne contient pas de subdivision de K_5 ou $K_{3,3}$.

Dual d'un graphe planaire

Si à partir d'un graphe planaire topologique G, on définit un graphe G^* tel que :

- à chaque face f de G, on fait correspondre un sommet f^* de G^*
- à chaque arête a de G, on fait correspondre une arête a^* de G^*
- deux sommets f^* et g^* sont reliées par une arête a^* de G^* si et seulement si les faces correspondantes f et g sont séparées par l'arête a de G.

alors on dit G^* est le *dual* de G (figure 3.6). Inversement, G est le dual de G^*.

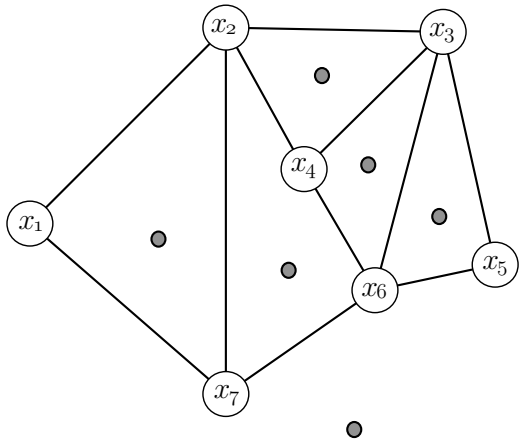


FIG. 3.6 – Graphe dual (en gris) du graphe planaire topologique de la figure 3.5(b).

Diagrammes de Voronoï et triangulation de Delaunay

Soit un espace M, et un ensemble S de points p choisis dans M que l'on va appeler *sites*. On associe à chaque p une région qui correspond à l'ensemble de tous les points $x \in M$ qui sont plus proches de p que de tout autre site de S. Un ensemble de régions ainsi construites est un diagramme de Voronoï. Nous donnons ci-dessous une définition formelle de tels diagrammes inspirée de Aurenhammer et al. (2000)[14].

Notons $d(x, y)$ la distance euclidienne entre deux points de M, et $\overline{A}$ la fermeture d'un ensemble A.

Appelons $D(p, q)$ le demi-plan défini par

$$p, q \in S; D(p, q) = \{x | d(p, x) < d(q, x)\} \tag{3.5}$$

Appelons $B(p, q)$ la ligne de séparation entre les demi-plans $D(p, q)$ et $D(q, p)$ définie par

$$p, q \in S; B(p, q) = \{x | d(p, x) = d(q, x)\} \tag{3.6}$$

Nous appellerons $VR(p, S)$ la *région de Voronoï* de p par rapport à S définie par

$$VR(p, S) = \bigcap_{q \in S, q \neq p} D(p, q) \tag{3.7}$$

La définition de régions de Voronoï à l'aide de demi-plans et des lignes qui les séparent est illustrée par la figure 3.7.

Enfin nous appellerons $V(S)$ le *diagramme de Voronoï* de S défini par

$$V(S) = \bigcup_{p, q \in S, q \neq p} \overline{VR(p, S)} \cap \overline{VR(q, S)} \tag{3.8}$$

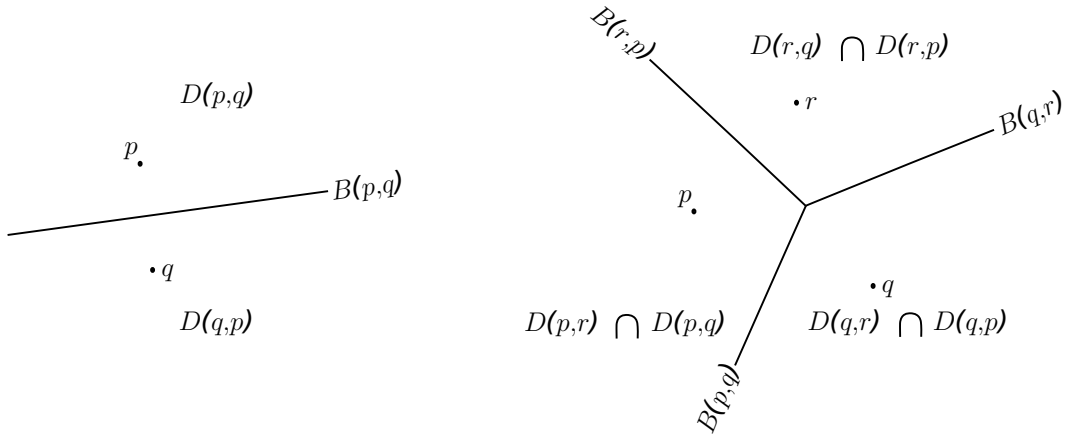


FIG. 3.7 – Illustration des éléments utilisés pour définir des régions de Voronoï

Une frontière commune entre deux régions de Voronoï est une *arête de Voronoï* et le point d'intersection de plusieurs arêtes de Voronoï est un *sommet de Voronoï*. Chaque sommet de Voronoï réunit en général 3 arêtes, et davantage dans des cas particuliers. On peut voir un diagramme de Voronoï comme un graphe qui comporte autant de faces qu'il y a de sites dans S. Il est important de noter que certaines de ces faces ont des arêtes infinies. Si l'espace M est de dimension 2, les faces sont des polygones convexes adjacents. Il est d'ailleurs pratique de limiter le diagramme à une boite englobante pour n'avoir que des polygones fermés (figure 3.8(a)).

Le graphe dual d'un diagramme de Voronoï est appellé *tesselation de Delaunay*. Dans le cas ou M est un espace de dimension deux, la tesselation de Delaunay est une triangulation (figure 3.8(b)).

Les triangulations de Delaunay possèdent plusieurs propriétés qui rendent leur usage intéressant. Il existe par exemple plusieurs façons de construire une triangulation à partir d'un ensemble de sites, celle de Delaunay maximise l'angle du plus petit de ses triangles. Les frontières des facettes extérieures de la triangulation constituent l'enveloppe convexe de l'ensemble des sites considérés.

Arbres

Un graphe connexe qui ne contient pas de cycle est appelé un *arbre*. Tout couple de sommets d'un arbre est relié par une chaîne et une seule.

Les sommets de degré 1 d'un arbre sont appelés *feuilles*.

Un graphe sans cycle comportant plusieurs composantes connexes est une *forêt*.

Dans un graphe orienté, on appelle *racine* un sommet a tel que tout autre sommet du graphe puisse être atteint à partir d'un chemin partant de a. Une racine n'existe pas toujours. Un arbre muni d'une racine est une *arborescence* (figure 3.9 (a)).

Un *arbre couvrant* d'un graphe G est un sous-graphe couvrant de G qui est un arbre.

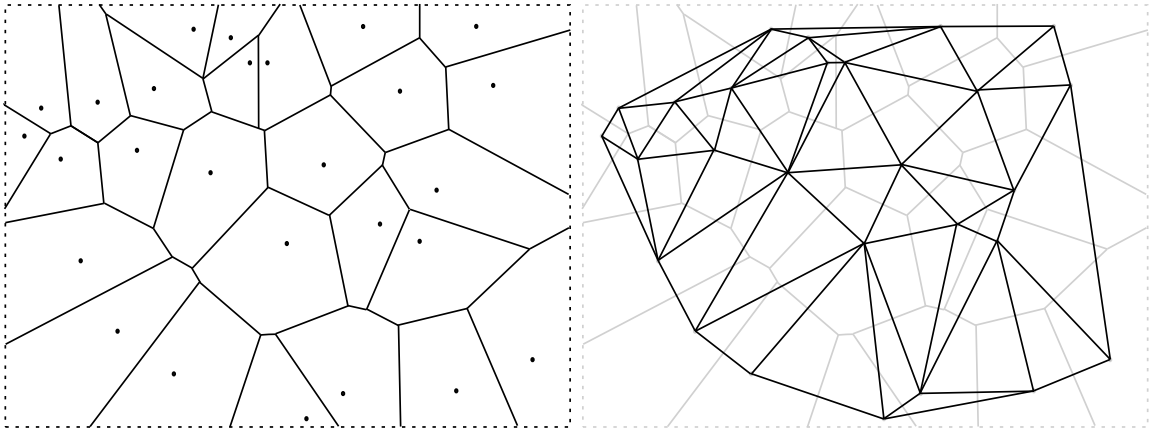


FIG. 3.8 – Diagramme de Voronoï (a) et triangulation de Delaunay duale de ce diagramme (b)

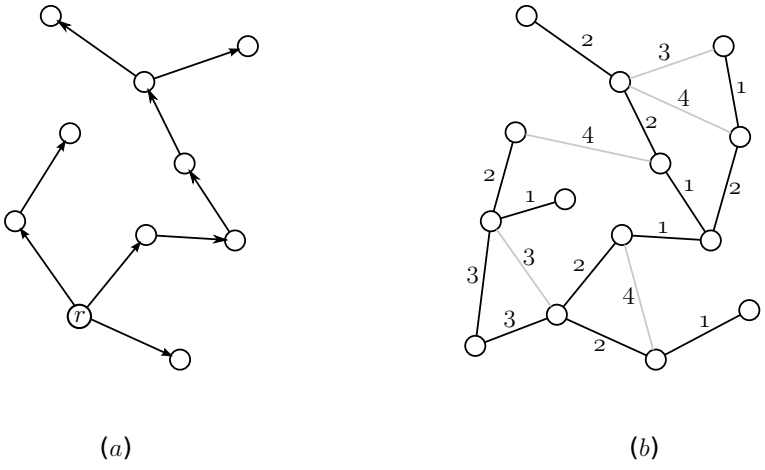


FIG. 3.9 – (a) Arborescence avec une racine r, et (b) arbre couvrant minimal d'un graphe valué

Tout graphe connexe admet au moins un arbre couvrant. Si G est un graphe valué, on peut chercher l'arbre couvrant dont la somme des valeurs associées aux arêtes est la plus petite possible ; on appelle un tel arbre *arbre couvrant minimal* de G (figure 3.9 (b)).

3.2.3 Hypergraphes

La définition habituelle des graphes porte sur des paires ou des couples de sommets. Mais on peut avoir besoin de définir des arêtes qui relient entre eux un nombre quelconque de sommets. C'est ce que C. Berge a appelé des *hypergraphes* [18] :

Soit $X = \{x_1, x_2, \ldots, x_n\}$ un ensemble fini, et $\mathcal{E} = (E_i / i \in I)$ une famille de parties de X. On dira que $\mathcal{E}$ constitue un hypergraphe sur X si l'on a :

1. $E_i \neq \emptyset$ $(i \in I)$
2. $\bigcup_{i \in I} E_i = X$

Les éléments de X sont les sommets et les ensembles E_i sont les arêtes. On représente en général les arêtes reliant plus de deux sommets par un trait qui les entoure (figure 3.10(a)).

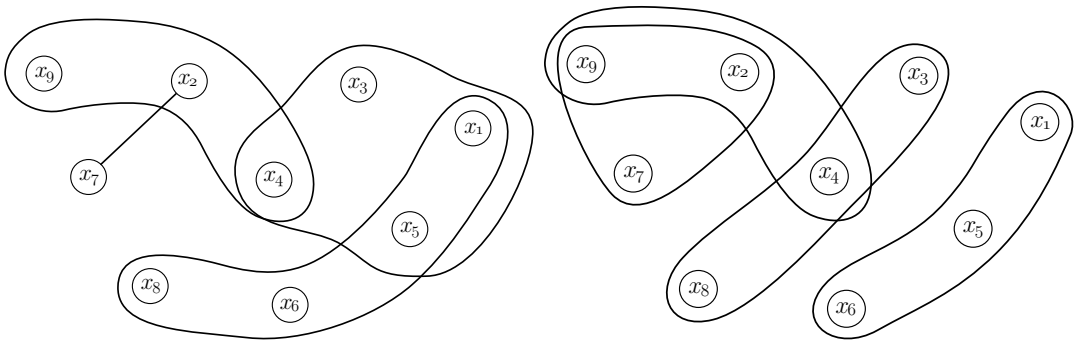


FIG. 3.10 – Hypergraphe (a) et hypergraphe uniforme de rang 3 (b)

Le *rang* d'un hypergraphe H correspond au plus grand nombre de sommets que l'on peut trouver dans une arête de H. Cela peut s'écrire comme une fonction $r(S)$ qui fait correspondre à tout sous-ensemble S de X le nombre entier positif :

$$r(S) = \max_i |S \cap E_i| \tag{3.9}$$

Si pour tout i, on a $|E_i| = r(X)$, c'est à dire que toutes les arêtes ont le même nombre de sommets, alors on dit que l'hypergraphe est *uniforme* (figure 3.10(b)).

3.3 Modélisation de dynamiques spatiales à l'aide de graphes

La conception d'une forme de modélisation passe par un certain nombre de définitions, de choix, et de questions qui sont présentées dans cette section. Nous reprenons les concepts de système et de modèle tels qu'ils sont définis dans la section 1.2 : un système est constitué d'entités qui interagissent les unes avec les autres, et un modèle est une représentation logique d'un système. La construction d'un modèle passe par le choix de certaines variables observables, et de la façon dont on peut décrire comment l'état de ces variables évolue dans le temps.

Nous avons choisi de distinguer trois niveaux dans la spécification d'un modèle :

Le niveau des individus Il s'agit de décrire les différents types d'entités que l'on va manipuler : leurs propriétés caractéristiques, ce qui constitue leur état, et les règles qui régissent leur comportement individuel.

Le niveau des interactions Que se passe-t-il lorsqu'une entité de type A interagit avec une entité de type B ? Y a-t-il transfert de matière, ou d'énergie de l'une vers l'autre ? Echangent-elles des informations ? Comment leurs états peuvent-ils se trouver modifiés à la suite de cette interaction ? Dans quelle conditions une entité A peut-elle intervenir dans plusieurs sortes d'interactions différentes et quel rôle joue-t-elle dans chaque cas ? Ce sont ces types de questions qui sont traitées à ce niveau. Nous sommes donc à la charnière entre l'individu et le système. C'est à ce niveau que nous faisons appel au concept de graphe et où nous donnons notre définition formelle du concept de *graphe d'interaction*.

Le niveau du système et de sa dynamique C'est un niveau plus opérationnel où l'on décrit l'enchaînement des opérations d'interaction, et d'évolution du système. On a accès à l'ensemble des composants d'un modèle et l'on peut les manipuler pour construire des structures de graphes d'interactions et séquencer dans le temps les opérations qui vont les faire évoluer.

Nous proposons dans cette section une description formelle de chacun de ces niveaux, et des éléments nécessaires à leur construction. Le niveau des individus concerne les concepts de propriété et d'entité (3.3.1) ; le niveau des interactions regroupe les concepts de rôle, d'interaction et de différentes fonctions associées (3.3.2), mais aussi les relations (3.3.3) et les graphes d'interaction (3.3.3) ; enfin le niveau du système et de sa dynamique est représenté par le concept de scénario (3.3.8).

3.3.1 Entité

Propriété

Une propriété représente une variable du système que l'on cherche à modéliser.

L'ensemble des valeurs que peut prendre une propriété constitue son *domaine*. Par exemple le domaine D d'une propriété de type booléenne est $D = \{vrai, faux\}$; pour une propriété représentant une température en degrés Kelvin, le domaine est l'ensemble des

nombres réels positifs $D = \mathbb{R}^+$.

La valeur prise par une propriété à un moment donné constitue son *état*. Certaines propriétés sont amenées à évoluer au cours du temps. Nous ne considérons ici que les évolutions par pas de temps discrets.

Pour rendre compte de ces aspects dynamiques nous adopterons les notations suivantes :

α_n représente l'état d'une propriété α au pas de temps n.

α_{n-1} représente l'état d'une propriété α au pas précédent le pas n et

α_{n+1} représente l'état d'une propriété α au pas suivant le pas n.

α_m^n représente la succession d'états pris par une propriété α entre les pas m et n. Dans ce cas, si D_α est le domaine de α, α_m^n est un k-uplet ($k = n - m + 1$) dont le domaine est le produit cartésien D_α^k

$\vec{\alpha}$ représente l'ensemble de l'historique des états pris par une propriété α depuis sa création jusqu'à l'état actuel. Si le dernier état est n, $\vec{\alpha}$ est donc une écriture concise de α_0^n. On utilisera la même forme de notation pour représenter le produit cartésien des domaines de tout l'historique de α : $\overrightarrow{D_\alpha} = \underbrace{D_\alpha \times \ldots \times D_\alpha}_{n+1 \ fois}$

Fonction de transition élémentaire

Une fonction f décrivant comment une propriété passe d'un état n à un état $n+1$ est appelée *fonction de transition élémentaire*. Ces fonctions de transition peuvent prendre diverses formes selon le nombre de propriétés, et la profondeur historique utilisés pour effectuer le calcul du nouvel état.

Voici à titre d'exemple trois fonctions f, g et h susceptibles d'affecter l'état d'une propriété α :

$$\begin{array}{ll} f : D_\alpha \mapsto D_\alpha, & \alpha_{n+1} = f(\alpha_n) \\ g : D_\alpha \times D_\beta \mapsto D_\alpha, & \alpha_{n+1} = g(\alpha_n, \beta_n) \\ h : \overrightarrow{D_\alpha} \times D_\gamma \mapsto D_\alpha, & \alpha_{n+1} = h(\vec{\alpha}, \gamma_n) \end{array} \tag{3.10}$$

f n'utilise que l'état actuel de α. g a besoin de l'état actuel de α et d'une autre propriété β. Enfin h fait usage de l'historique de tous les états passés de α et de l'état actuel d'une propriété γ.

Entité

Une *entité* est un élément de modèle défini par un p-uplet de propriétés dont les domaines peuvent être différents.

Soit un ensemble d'entités X et une entité $x \in X$, $x = (\underbrace{\alpha, \beta, \ldots}_{p \ propriétés})$.

Dans cet exemple $\alpha, \beta, \ldots$ sont les propriétés de x et $\underbrace{D_\alpha, D_\beta, \ldots}_{p \text{ domaines}}$ leurs domaines de valeurs respectifs.

L'état d'une entité x est constitué des valeurs de ses propriétés à un moment donné. Cet état est donc un élément du produit cartésien des domaines des propriétés de l'entité. Nous noterons ce produit cartésien $D_x = \underbrace{D_\alpha \times D_\beta \times \ldots}_{p \text{ domaines}}$

De manière similaire aux notations relatives à la dynamique des propriétés, nous adoptons les notations suivantes pour rendre compte des aspects dynamiques des entités :

x_n représente l'état de l'entité x au pas de temps n : $x_n = (\alpha_n, \beta_n, \ldots)$ avec $x_n \in D_x$

x_{n-1} et x_{n+1} représentent respectivement les états de l'entité x aux pas de temps précédent le pas n et suivant le pas n.

x_m^n représente la succession d'états de l'entité x entre les pas temps m et n : $x_m^n = (\alpha_m^n, \beta_m^n, \ldots)$ et $x^n \in D_x^{n-m+1}$

$\vec{x}$ représente l'ensemble de l'historique des états pris par l'entité x depuis sa création jusqu'à l'état actuel et $\vec{x} \in \overrightarrow{D_x}$

Equivalences d'entités

On dira que deux entités x et y sont équivalentes (noté $x \Leftrightarrow y$) si $D_x = D_y$. Les caractères réflexif, symétrique et transitif de cette relation sont évidents, l'ensemble des entités ayant un domaine commun constitue ainsi une classe d'équivalence.

Dans le cas où l'équivalence n'est que partielle, par exemple avec $x = (\alpha, \beta)$ et $y = (\alpha, \beta, \gamma)$, celle-ci peut être mise à profit pour doter x et y d'un même comportement comme nous le verrons à travers la notion de *rôle* dans la section suivante.

Pour éviter toute ambiguïté, dans les cas où plusieurs entités possèdent des propriétés de même nom (comme c'est le cas pour x et y ci-dessus) nous adopterons une notation dans laquelle le nom de la propriété est préfixé par le nom de l'entité à laquelle elle appartient. Par exemple $x : \alpha$ et $y : \alpha$ sont respectivement les propriétés α des entités x et y.

Fonctions sur une entité

Pour mesurer l'état d'une ou plusieurs propriétés d'une entité, ou pour faire évoluer cet état, on peut définir des fonctions portant sur les propriétés de cette entité. Ces fonctions seront associées à toutes les entités d'une classe d'équivalence :

Fonctions de consultation Ce sont les fonctions qui permettent de lire l'état d'une entité sans modifier cet état. Dans le cas général, les fonctions de consultation peuvent

porter sur l'ensemble de l'historique des états successifs d'une entité. Soit x une entité et D_m le domaine des valeurs possibles du résultat, une fonction de consultation f_c sera définie par

$$f_c : \overrightarrow{D_x} \mapsto D_m, \quad \delta = f_c(\vec{x}) \tag{3.11}$$

Dans les cas où $D_m = \{vrai, faux\}$, f_c peut être considéré comme une fonction de test.

Fonctions de Transition Ces fonctions sont constituées d'une ou plusieurs fonctions de transition élémentaires portant sur les propriétés d'une entité. Elles permettent d'affecter l'état d'une entité :

$$f_t : \overrightarrow{D_x} \mapsto D_x, \quad x_{n+1} = f_t(\vec{x}) \tag{3.12}$$

3.3.2 Rôle

Lorsque l'on construit un modèle à des fins de simulation, il faut donner une description de la nature des interactions qui ont lieu dans le système que l'on cherche à modéliser. Dans les domaines d'application qui nous intéressent, il serait dans bien des cas fastidieux de décrire une à une chaque interaction individuelle intervenant dans le modèle. Nous allons plutôt chercher à décrire des interactions entre des classes d'équivalence d'entités. Par exemple on va décrire comment une espèce donnée de plantes échange du carbone avec l'atmosphère et appliquer cette description à toutes les plantes de cette espèce présentes dans le modèle. La fonction décrivant la nature de l'interaction pourra être applicable à toutes les entités qui possèdent les propriétés impliquées dans cette interaction. Cela signifie que l'interaction ainsi définie pourrait éventuellement être appliquée sur d'autres espèces de plantes pour peu que celles-ci possèdent des propriétés équivalentes à l'espèce qui a servi de modèle pour définir l'interaction.

D'une manière plus générique, considérons que l'ensemble des propriétés impliquées dans une fonction d'interaction va représenter le *rôle* que les entités peuvent jouer lorsqu'elles interagissent.

Un rôle est défini de la même manière qu'une entité, comme un p-uplet de propriétés. Un rôle reste cependant une définition abstraite servant de modèle aux entités pour définir des fonctions de manière générique.

Soit a un rôle défini par : $a = (\alpha 1, \alpha 2, \ldots, \alpha p)$ avec $\alpha 1 \in D_{\alpha 1}, \alpha 2 \in D_{\alpha 2}, \ldots, \alpha p \in D_{\alpha p}$

On dira qu'une entité x peut jouer le rôle a si l'on a

$$\forall \alpha \in a, \exists \beta \in x \text{ tel que } D_\alpha = D_\beta \tag{3.13}$$

En d'autres termes, x peut jouer le rôle a à condition que $D_a \subset D_x$.

On voit à partir de cette définition que plusieurs entités pas forcément équivalentes peuvent jouer un même rôle :

Soient x et y deux entités telles que

$$D_x \cap D_y \neq \emptyset \tag{3.14}$$

supposons qu'il existe un rôle a pour lequel on ait :

$$D_a \subset (D_x \cap D_y) \tag{3.15}$$

alors x et y peuvent jouer le même rôle a.

Les entités x et y ne sont en effet pas équivalentes de manière absolue mais du point de vue du rôle a elles appartiennent à une même classe d'équivalence définie par D_a : celle des entités pouvant jouer le rôle a.

Les rôles sont des concepts abstraits, et en toute rigueur on ne devrait pas pouvoir parler de l'état d'un rôle. Pour éviter d'alourdir les notations, il sera cependant pratique de reporter sur les rôles les conventions de notations adoptées pour les entités. Soit plus précisément :

a_n représente l'état d'une entité jouant le rôle a au pas de temps n.

a_{n-1} et a_{n+1} représentent respectivement les états d'une entité jouant le rôle a aux pas de temps précédent le pas n et suivant le pas n.

a_m^n représente la succession d'états d'une entité jouant le rôle a entre les pas temps m et n

$\vec{a}$ représente l'ensemble de l'historique des états pris par une entité jouant le rôle a depuis sa création jusqu'à l'état actuel.

Comme pour les entités, nous pouvons définir des fonctions portant sur la classe d'équivalence représentée par un rôle. Mais nous allons aussi définir des fonctions impliquant plusieurs rôles en même temps : les fonctions d'interaction.

Fonctions portant sur un rôle

Il s'agit des fonctions de consultation et de transition dont la définition (donnée ici pour un rôle a) est identique à celle que l'on a donné sur les entités :

Fonctions de consultation

$$f_c : \overrightarrow{D_a} \mapsto D_m, \quad \delta = f_c(\vec{a}) \tag{3.16}$$

Fonctions de transition

$$f_t : \overrightarrow{D_a} \mapsto D_a, \quad a_{n+1} = f_t(\vec{a}) \tag{3.17}$$

Fonctions portant sur plusieurs rôles et interaction

Fonctions de consultation Définir des fonctions de consultation portant sur plusieurs rôles permet d'abord d'accéder à une information qui n'est pertinente ou disponible que lorsque des entités sont prises en considération ensemble, chacune correspondant à l'un des rôles. Un exemple peut être le calcul d'une distance entre deux entités : ce calcul n'a pas d'utilité pour une entité seule, et il ne modifie en rien l'état des entités considérées.

Soient k rôles $a, b, \ldots$:

$$f_c : D_a \times D_b \times \ldots \mapsto D_m, \quad \delta = f_c(\vec{a}, \vec{b}, \ldots) \text{ avec } \delta \in D_m \tag{3.18}$$

Les fonctions f_c ainsi définies calculent un résultat δ à partir de l'état de propriétés pouvant provenir des k rôles que l'on s'est donné.

Fonctions d'interaction Pour qu'il y ait interaction entre une entité x et une ou plusieurs autres entités, il faut qu'une fonction de transition affectant au moins une propriété de x fasse intervenir au moins une propriété d'une autre entité que x dans le calcul du nouvel état.

Il s'agit ici de définir la nature des interactions entre entités de manière générique, indépendamment de la structure, c'est à dire sans précision aucune sur les liens qui peuvent exister entre les entités. C'est la principale raison pour laquelle les fonctions de transition sont définies sur des rôles et non directement sur des entités.

Au moins deux rôles sont donc nécessaires pour que la définition d'une fonction d'interaction soit possible. Les interactions portant sur deux rôles seulement sont d'ailleurs ce que l'on va probablement rencontrer dans une grande majorité des cas de modélisation. Comme dans les cas précédents, nous allons donner ici une définition aussi générale que possible, portant non seulement sur un nombre k fini de rôles, mais aussi en prenant en compte l'ensemble des états passés de chaque propriété.

Soient k rôles $a = (\alpha 1, \ldots, \alpha p), b = (\beta 1, \ldots, \beta q), c = \ldots$

La fonction $f : D_a \times D_b \times \ldots \mapsto D_a \times D_b \times \ldots$ qui fait passer a, b, ... d'un état n à un état suivant $n+1$ en impliquant l'ensemble des propriétés des rôles est appelée *fonction d'interaction* :

$$(a_{n+1}, b_{n+1}, \ldots) = f(\vec{a}, \vec{b}, \ldots) \tag{3.19}$$

Cette fonction d'interaction correspond à un ensemble d'opérations de transition décrivant l'évolution de chaque propriété des k rôles $a, b, \ldots$ à partir des états passés

de une ou plusieurs autres propriétés :

$$\begin{cases} \alpha 1_{n+1} = f_{\alpha 1}(\vec{\alpha 1}, \ldots, \vec{\alpha p}, \vec{\beta 1}, \ldots) \\ \alpha 2_{n+1} = f_{\alpha 2}(\vec{\alpha 2}, \ldots, \vec{\alpha p}, \vec{\beta 1}, \ldots) \\ \ldots \\ \beta 1_{n+1} = f_{\beta 1}(\vec{\beta 1}, \ldots, \vec{\beta q}, \vec{\alpha 1}, \ldots) \\ \beta 2_{n+1} = f_{\beta 2}(\vec{\beta 2}, \ldots, \vec{\beta q}, \vec{\alpha 1}, \ldots) \\ \ldots \end{cases} \tag{3.20}$$

L'opération qui permet de passer à l'état $(a_{n+1}, b_{n+1}, \ldots)$ à travers la fonction f revient à appliquer simultanément l'ensemble des opérations élémentaires $f_{\alpha 1}, \ldots, f_{\alpha p}, f_{\beta 1}, \ldots, f_{\beta q}, \ldots$.
Nous verrons (en 3.3.3 : *Situations d'affectations multiples et simultanées*) que cette simultanéité peut amener à des situations d'indétermination, selon la structure des liens entre entités sur laquelle on cherchera à appliquer une fonction d'interaction.

3.3.3 Relation et graphe d'interactions

Relation

Les différentes formes de fonctions portant sur des couples d'entités (et parfois davantage) ont été définies à partir de rôles. Lorsque lors de la conception d'un modèle, on définit une fonction à partir de deux rôles a et b par exemple, c'est pour s'assurer que cette fonction pourra accéder aux propriétés de a et de b, soit pour effectuer des calculs, soit pour modifier l'état de certaines de ces propriétés.

Une autre façon de concevoir les choses consiste à définir toutes les fonctions que l'on puisse appliquer à des rôles donnés. Il est tout à fait possible en effet que certains ensembles de propriétés soient appropriés pour être mis en jeu dans plus d'une fonction d'interaction (ou de consultation).

C'est à partir du choix de certains rôles et de l'ensemble des fonctions portant sur ces rôles que nous pouvons définir le concept de *Relation*.

Soit A un k-uplet de rôles $A = (a1, \ldots, ak)$

Notons F_c l'ensemble des fonctions de consultation portant sur ces rôles, F_t l'ensemble des fonctions de transition (sur chaque rôle pris séparément) et F_i l'ensemble des fonctions d'interaction. Appelons F l'ensemble de toutes les fonctions d'un modèle portant sur les rôles de A : $F = F_c \cup F_t \cup F_i$

Nous appelons *Relation* (noté $\Re$) le couple :

$$\Re = (A, F) \tag{3.21}$$

Une relation définit la nature des actions pouvant intervenir dans un modèle à partir des rôles que des entités peuvent jouer lorsqu'elles interagissent ou lorsque l'on fait appel

à des fonctions de consultation. Une relation ne porte donc pas directement sur les entités d'un modèle mais définit des contraintes (les propriétés de chaque rôle de l'ensemble A) permettant de savoir sur quelles classes d'équivalence d'entités les diverses fonctions pourront être appliquées.

Graphe d'interaction

Pour préciser concrètement quelles sont les entités qui vont effectivement être impliquées dans des interactions, on peut construire une ou plusieurs structures de graphes qui vont porter les liens entre entités. C'est en associant une structure de graphe et une relation que nous définissons le concept de graphe d'interaction :

Nous appelons *Graphe d'interaction* le triplet :

$$\Gamma = (\Re, X, U) \tag{3.22}$$

avec
$\Re = (A, F)$ une relation définie sur k rôles avec $A = (a1, \ldots, ak)$
$X = \{x1, \ldots, xn\}$ l'ensemble des entités d'un modèle.
$U = (u1, u2, \ldots, um)$ une famille dont les éléments sont issus du produit cartésien :
$X^k = \{(x1, x2, \ldots, xk)/x1 \in X, x2 \in X, \ldots, xk \in X\}$ et $D_{a1} \subset D_{x1}, D_{a2} \subset D_{x2}, \ldots, D_{ak} \subset D_{xk}$

La relation $\Re$ porte une description de la nature des interactions et du rôle joué par des entités, et le graphe donné par le couple (X, U) représente la structure sur laquelle ces interactions pourront avoir lieu.

Cette définition de graphe d'interaction représente le cas général pour lequel on ne précise rien sur la valeur de k. Dans la grande majorité des cas, on peut considérer que l'on aura $k = 2$ et (X, U) sera en fait un graphe biparti. Pour les cas où $k > 2$, (X, U) constituera une structure d'hypergraphe uniforme de rang k. Dans la suite de ce document il est principalement question de graphes pour lesquels $k = 2$, la valeur de k n'est donc précisée que lorsque l'on traite d'hypergraphes.

Un graphe d'interaction peut être représenté sous la forme d'un graphe orienté avec les éléments de X (les entités) comme sommets, et les éléments de U comme arcs. Le sens de la flèche indique quel est le rôle joué par chaque extrémité de l'arc. Il est important de noter que le sens de la flèche n'a aucun rapport avec le sens d'un éventuel transfert d'information, de matière ou de quoi que ce soit entre les entités. Le fait que l'on fasse appel à une représentation par un graphe orienté vient du caractère non symétrique de la définition des relations : chaque rôle est différent et peut être utilisé de manière différencié par les fonctions d'interaction.

Prenons par exemple une relation basée sur le couple de rôles (a, b) et appliquons cette relation à un ensemble d'entités $X = \{x; y; z; v\}$. La figure 3.11(a) montre simplement

à quelle extrémité de la flèche, chaque rôle est associé. La figure 3.11(b) représente un graphe pour lequel $U = (u1)$ avec $u_1 = (x, y)$. On peut construire ce graphe à condition que x possède les bonnes propriétés pour jouer le rôle a et y possède les bonnes propriétés pour jouer le rôle b de la relation que l'on s'est donné.

Sur la figure 3.11(c) nous avons un graphe pour lequel $U = (u1, u2, u3)$ et l'on voit que $u1$ et $u2$ relient les mêmes entités x et y mais en leur faisant jouer des rôles inversés. Dans ce cas, les deux entités x et y doivent toutes deux posséder à la fois des propriétés compatibles avec les rôles a et b pour que de telles interactions puissent être définies. L'arc $u3$ fait partie du même graphe et constitue une seconde composante connexe de ce graphe.

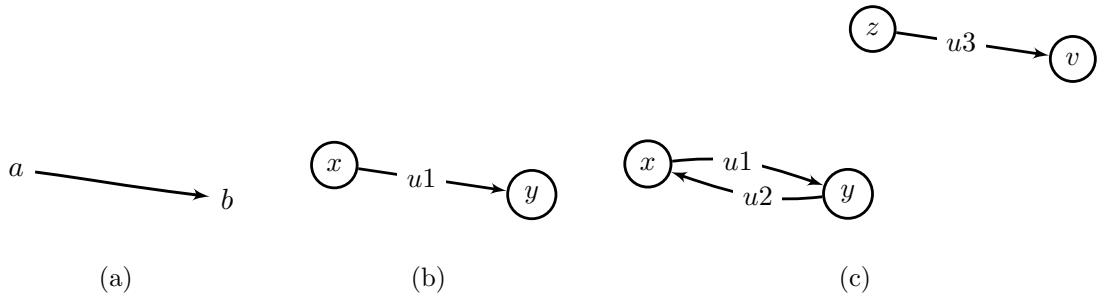


FIG. 3.11 – Représentation de graphe d'interactions par des arcs orientés

Puisque $\Re$ est une construction abstraite, l'évolution dans le temps d'un graphe d'interaction $\Gamma = (\Re, X, U)$ n'affecte que la partie "graphe" proprement dite de Γ, c'est à dire le couple (X, U). L'état de Γ à un pas de temps n peut être exprimé par : $\Gamma_n = (\Re, X_n, U_n)$ où X_n représente l'ensemble des états des entités de X et U_n représente l'état de la structure du graphe à ce moment là.

Le fait de disposer d'un graphe dont les sommets sont des entités possédant des propriétés connues (grâce aux rôles de $\Re$) nous permet d'utiliser ces fonctions sur toutes les entités du graphe. Nous avons voulu faire en sorte que le modélisateur puisse agir sur l'ensemble d'un graphe Γ d'une manière déclarative, c'est à dire sans se préoccuper de l'ordre dans lequel les éléments du graphe seront traités.

Trois possibilités se présentent au modélisateur pour manipuler un graphe d'interaction, que ce soit pour consulter son état ou pour le faire évoluer :

1. Utiliser des fonctions définies sur un seul rôle de $\Re$, et appliquer ces fonctions aux entités de X pouvant jouer ce rôle. On travaille dans ce cas directement sur un sous-ensemble d'entités du graphe qu'elles soient reliées ou non à un arc de U.
2. Utiliser des fonctions d'interaction ou de consultation sur tous les rôles de $\Re$ qui, en s'appliquant aux arcs du graphe, vont donner accès aux entités reliées les unes aux autres.
3. Agir sur la structure même du graphe, c'est à dire sur le contenu de U pour l'analyser, ou le modifier.

Chacune de ces trois options est détaillée dans les sections qui suivent.

3.3.4 Agir sur les entités indépendamment de la structure de Γ

Soit un graphe d'interaction $\Gamma = (\Re, X, U)$ et $\Re = (F, A)$. Choisissons une fonction $f_a \in F$ définie sur un seul rôle $a \in A$. La définition de cette fonction spécifie une action sur une entité quelconque qui possède les propriétés de a. Lorsque l'on parle d'appliquer f_a sur tout le graphe, c'est une facilité de langage pour exprimer le fait que l'on va appliquer cette fonction à toutes les entités de $X_a \subset X$ et, si cette fonction produit un résultat, obtenir l'ensemble des résultats de ces applications. Notons au passage que X_a inclut les entités isolées (s'il y en a) qui sont contenues dans X mais qui ne sont pas reliées à d'autres par un arc de U.

Consultation

Dans le cas où f_a est une fonction de consultation, cette fonction produit un résultat δ ayant D_m pour domaine. Son application au graphe va produire l'ensemble des résultats Δ_X défini par :

$$\Delta_X = \{\delta \in D_m / \forall x \in X_a, \delta = f_a(\vec{x})\} \tag{3.23}$$

Nous aurons $card(\Delta_X) = card(X_a)$ et il n'y a pas à priori de relation d'ordre sur les éléments de Δ_X découlant de cette opération : la consultation des éléments de X_a est considérée comme simultanée.

Sélection

Lorsque la fonction choisie est une fonction de test, c'est à dire un cas particulier des fonctions de consultation qui renvoie un résultat booléen ($Dm = \{vrai; faux\}$), on peut s'en servir pour effectuer une sélection d'entités. Le résultat X'_a est alors un sous-ensemble de X_a dont les éléments vérifient la proposition logique exprimée par la fonction f_a :

$$X'_a = \{x \in X_a / f_a(\vec{x}) = vrai\} \tag{3.24}$$

Remarquons que $card(X'_a)$ peut varier à chaque pas de temps et X'_a peut parfois être un ensemble vide.

Transition

Si la fonction f_a est une fonction de transition, son effet sera d'affecter l'état des éléments de X_a auxquels elle sera appliquée. Son application sur le graphe correspond à l'action suivante :

$$X_a = \{x_{n+1} / x_{n+1} = f_a(\vec{x})\} \tag{3.25}$$

On considère que cette opération change l'état de tous les éléments de X_a de façon simultanée.

3.3.5 Agir sur les sommets des arcs de Γ

Soit un graphe d'interaction $\Gamma = (\Re, X, U)$ et $\Re = (F, A)$. Choisissons une fonction $f \in F$ définie sur le k-uplet de tous les rôles de A. La définition de cette fonction spécifie une action sur les entités d'un k-uplet $u \in U$. Lorsque l'on parle d'appliquer f sur tout le graphe, cela signifie que l'on va appliquer cette fonction à chaque arc u du graphe et, si cette fonction produit un résultat, obtenir l'ensemble des résultats de ces applications. Il faut remarquer que nous n'aurons donc accès qu'aux entités reliées par des arcs du graphe, les entités isolées seront ignorées par ces opérations.

Consultation

Dans le cas où f est une fonction de consultation, cette fonction produit un résultat δ ayant D_m pour domaine. Son application au graphe va produire l'ensemble des résultats Δ_U défini par :

$$\Delta_U = \{\delta \in D_m / \forall u = (x1, \ldots, xk) \in U, \delta = f(\vec{x1}, \ldots, \vec{xk})\} \tag{3.26}$$

Nous aurons $card(\Delta_U) = card(U)$ et il n'y a pas à priori de relation d'ordre sur les éléments de Δ_U découlant de cette opération : la consultation des éléments de U est considérée comme simultanée.

Sélection

Lorsque la fonction choisie est une fonction de test, c'est à dire un cas particulier des fonctions de consultation qui renvoie un résultat booléen ($Dm = \{vrai; faux\}$), on peut s'en servir pour effectuer une sélection d'arcs. Le résultat U' est alors un sous-ensemble de U dont les éléments vérifient la proposition logique exprimée par la fonction f :

$$U' = \{u = (x1, \ldots, xk) \in U / f(\vec{x1}, \ldots, \vec{xk}) = vrai\} \tag{3.27}$$

$card(U')$ peut varier à chaque pas de temps et U' peut parfois être un ensemble vide.

Interaction

Appliquer une fonction d'interaction $f \in F$ à un graphe Γ signifie effectuer simultanément les opérations de transition de f sur les entités aux extrémités de chaque arc de U. C'est donc potentiellement l'ensemble des entités de Γ qui peut changer d'état par l'application d'une fonction d'interaction :

$$X = \{x \in X / \forall u = (x1_n, \ldots, xk_n) \in U, (x1_{n+1}, \ldots, xk_{n+1}) = f(\vec{x1}_n, \ldots, \vec{xk}_n)\} \tag{3.28}$$

Examinons concrètement, à l'aide d'un exemple, ce que cela signifie et les questions que ce caractère simultané peut soulever :

Soient les rôles $a = (\alpha, \gamma)$ et $b = (\beta)$ et une fonction d'interaction f définie sur ces rôles et contenant deux opérations de transition :

$$f(\alpha, \gamma, \beta) \Rightarrow \begin{cases} \alpha_{n+1} = \frac{1}{3}(2 \times \alpha_n + \beta_n) \\ \beta_{n+1} = \beta_n - \gamma_n \end{cases} \tag{3.29}$$

L'application de cette fonction f aux entités d'un arc (x, y) du graphe a donc pour effet de changer l'état de la propriété $x : \alpha$ de l'entité x jouant le rôle a et $y : \beta$ de l'entité y jouant le rôle b aux extrémités de cet arc.

Ainsi si l'on applique f au graphe de la figure 3.12(a) nous allons simultanément obtenir les changements d'états de la propriété α de chacune des entités x, y, u et w jouant le rôle a, ainsi que de la propriété β des entités y, z, v et w jouant le rôle b.

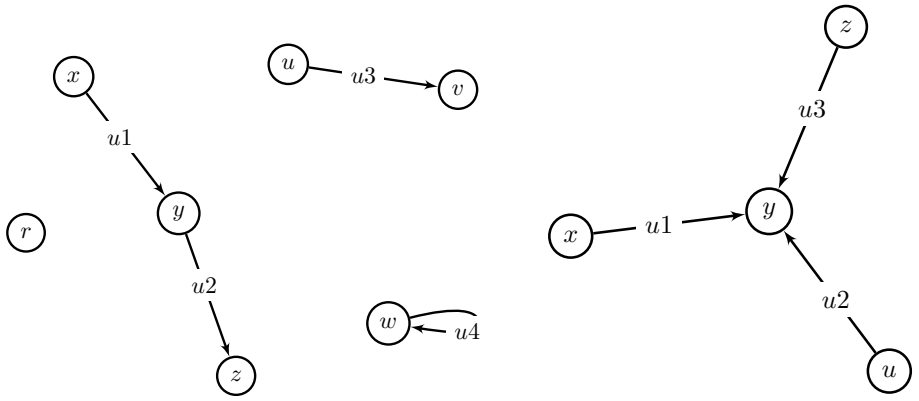
Voici le détail des changements d'états effectués dans cet exemple par l'application des opérations de transition sur chaque arc :

$$\begin{aligned}
\text{Sur } u_1 : f(x : \alpha, x : \gamma, y : \beta) &\Rightarrow \begin{cases} x : \alpha_{n+1} = \frac{1}{3}(2 \times x : \alpha_n + y : \beta_n) \\ y : \beta_{n+1} = y : \beta_n - x : \gamma_n \end{cases} \\
\text{Sur } u_2 : f(y : \alpha, y : \gamma, z : \beta) &\Rightarrow \begin{cases} y : \alpha_{n+1} = \frac{1}{3}(2 \times y : \alpha_n + z : \beta_n) \\ z : \beta_{n+1} = z : \beta_n - y : \gamma_n \end{cases} \\
\text{Sur } u_3 : f(u : \alpha, u : \gamma, v : \beta) &\Rightarrow \begin{cases} u : \alpha_{n+1} = \frac{1}{3}(2 \times u : \alpha_n + v : \beta_n) \\ v : \beta_{n+1} = v : \beta_n - u : \gamma_n \end{cases} \\
\text{Sur } u_4 : f(w : \alpha, w : \gamma, w : \beta) &\Rightarrow \begin{cases} w : \alpha_{n+1} = \frac{1}{3}(2 \times w : \alpha_n + w : \beta_n) \\ w : \beta_{n+1} = w : \beta_n - w : \gamma_n \end{cases}
\end{aligned} \tag{3.30}$$

On peut constater que les propriétés de chaque entité sont affectées de nouvelles valeurs de manière non ambiguë. Si par exemple l'on ne pouvait effectuer ces opérations que sur un seul arc à la fois, l'ordre dans lequel on traiterait les arcs n'aurait pas d'importance : le résultat serait le même.

Une fois ces opérations effectuées sur le graphe, on considère que toutes les entités du graphe sont dans l'état $n+1$. En conséquence, toutes leurs propriétés sont aussi passées à l'état $n+1$, y compris celles qui n'ont pas été explicitement affectées par les opérations de transition de f. Dans cet exemple, la propriété γ du rôle a est utilisée seulement en lecture par f. On considère dans ce cas que f a aussi effectué de manière implicite l'opération $\gamma_{n+1} = \gamma_n$ sur chacune des entités jouant le rôle a.

Le changement d'état inclut également les entités isolées comme l'entité r de la figure 3.12(a). De telles entités font partie du graphe (elles sont présentes dans X) mais ne sont reliées à aucune autre par un arc (on ne les trouve dans aucun élément de U). On considère que ces entités isolées sont passées à l'état $n+1$ en conservant leurs valeurs de propriétés inchangées.



(a) (b)

FIG. 3.12 – Exemples de graphes sur lesquels appliquer une fonction d'interaction

Situations d'affectations multiples et simultanées

Le caractère simultané de l'application des opérations de transition à l'ensemble des arcs d'un graphe peut dans certaines configurations conduire à des situations d'affectation multiple d'une même propriété. La valeur de cette propriété risque alors de se trouver indéterminée.

Ces configurations peuvent se ramener à deux sortes de situations : la première est liée au partage d'une même propriété par des rôles différents, la seconde est liée à une multiplicité d'arcs incidents sur une entité. Chacun de ces deux cas est décrit en détail ci-dessous :

Situation de partage de propriété Dans une définition de relation, il est possible qu'une même propriété soit présente dans plusieurs rôles :

Soient les propriétés α, β, γ et deux rôles a et b tels que : $a = (\alpha, \beta), b = (\alpha, \gamma)$. Si l'on définit une fonction d'interaction qui affecte la propriété α pour chacun de ces rôles :

$$f : \begin{cases} a : \alpha_{n+1} = f_{a:\alpha}(a : \alpha_n, a : \beta_n, b : \alpha_n, b : \gamma_n) \\ b : \alpha_{n+1} = f_{b:\alpha}(a : \alpha_n, a : \beta_n, b : \alpha_n, b : \gamma_n) \end{cases} \tag{3.31}$$

Il n'y a pas d'indétermination tant qu'aucune entité du graphe ne joue les deux rôles à la fois.

Mais si l'on applique une telle fonction par exemple sur le graphe de la figure 3.12(a), les entités y et w sont dans la situation de jouer en même temps les deux rôles a et b. La propriété α de ces entités va se trouver affectée simultanément par les deux opérateurs de transition $f_{a:\alpha}$ et $f_{b:\alpha}$. Il y a une indétermination pour la valeur de α dans ce cas.

Pour éviter de se retrouver dans une telle situation, on peut par exemple interdire qu'une fonction d'interaction affecte une même propriété partagée par deux rôles différents. Ou si l'on autorise de telles fonctions, on peut interdire de les appliquer sur un graphe dans lequel certaines propriétés jouent simultanément les rôles qui partagent une même propriété. On peut encore interdire à différents rôles de partager une même propriété.

Situation de multiplicité d'arcs incidents Soit $\Gamma = (\Re, X, U)$ un graphe d'interaction comprenant une relation définie sur deux rôles : $\Re = ((a, b), F)$.

Supposons que l'on se trouve dans une situation telle qu'une fonction d'interaction affecte une propriété α du rôle a :

$$\exists f \in F, \exists \alpha \in a, \exists \beta \in b, f : D_\alpha \times D_\beta \mapsto D_\alpha \tag{3.32}$$

En appliquant f au graphe, la propriété α de toute entité x jouant le rôle a sera affectée d'une valeur pour chacun des arcs incidents à l'extérieur de x. En conséquences, les entités x du graphe pour lesquelles on a $d_G^+(x) > 1$, la propriété α va se trouver en situation d'indétermination puisqu'elle sera affectée simultanément de plusieurs valeurs différentes.

D'une manière symétrique, on se trouvera aussi en situation d'indétermination si l'on applique une fonction g au graphe avec :

$$\begin{cases} \exists g \in F, \exists \alpha \in a, \exists \beta \in b, g : D_\alpha \times D_\beta \mapsto D_\beta \\ \exists (y, x) \in U \text{ et } d_\Gamma^-(x) > 1 \end{cases} \tag{3.33}$$

A titre d'exemple on peut appliquer la fonction d'interaction f définie dans la formule 3.29 au graphe de la figure 3.12(b). Le détail des opérations de changement d'état nous donne :

$$\begin{aligned} \text{Sur } u_1 : f(x : \alpha, x : \gamma, y : \beta) \Rightarrow & \begin{cases} x : \alpha_{n+1} = \frac{1}{3}(2 \times x : \alpha_n + y : \beta_n) \\ y : \beta_{n+1} = y : \beta_n - x : \gamma_n \end{cases} \\ \text{Sur } u_2 : f(z : \alpha, z : \gamma, y : \beta) \Rightarrow & \begin{cases} z : \alpha_{n+1} = \frac{1}{3}(2 \times z : \alpha_n + y : \beta_n) \\ y : \beta_{n+1} = y : \beta_n - z : \gamma_n \end{cases} \\ \text{Sur } u_3 : f(u : \alpha, u : \gamma, y : \beta) \Rightarrow & \begin{cases} u : \alpha_{n+1} = \frac{1}{3}(2 \times u : \alpha_n + y : \beta_n) \\ y : \beta_{n+1} = y : \beta_n - u : \gamma_n \end{cases} \end{aligned} \tag{3.34}$$

On constate que la propriété $y : \beta_{n+1}$ se trouve affectée trois fois, une fois pour chaque arc incident à l'entité y, et il n'y a aucune raison pour que ces trois valeurs soient identiques. Il y a là une indétermination pour le nouvel état de $y : \beta$ qu'il nous faudra chercher à lever.

Poser des contraintes pour éviter la situation de multiplicité d'arcs incidents est certainement possible dans certains cas de modélisation : il faut pour cela interdire à une entité d'avoir plus d'un arc incident à l'extérieur et plus d'un arc incident à l'intérieur. Une telle contrainte risque cependant d'être trop forte pour une majorité de modèles et c'est la

raison pour laquelle nous voulons proposer une solution moins restrictive, qui consiste à définir des fonctions chargées de lever l'indétermination lorsque la situation se présente : les opérateurs d'agrégation.

Opérateurs d'agrégation

Nous appellerons *opérateur d'agrégation* sur une propriété α une fonction h définie par :

$$h : D_\alpha^k \mapsto D_\alpha,\ k \in N \tag{3.35}$$

Un tel opérateur est conçu pour être utilisé lors de l'affectation d'une propriété dans les situations où il y a plusieurs valeurs "candidates" simultanées. L'opérateur d'agrégation traite l'ensemble de ces k valeurs candidates et produit une valeur unique qui est celle qui sera effectivement affectée à la propriété sur laquelle il est utilisé. La valeur retenue peut être l'une des valeurs candidates comme par exemple la plus grande, la plus petite, ou l'une d'entre elles prise au hasard. Mais il peut aussi s'agir d'une nouvelle valeur synthétisée à partir de l'ensemble des candidates, comme la moyenne ou la médiane.

Au delà de ces exemples simples, un modélisateur doit pouvoir définir des opérateurs d'agrégation adaptés à ses besoins. Dans une situation de multiplicité d'arcs incidents, les valeurs candidates sont liées aux entités voisines dans le graphe. Un opérateur d'agrégation peut dans ce cas être défini pour décrire comment une propriété change d'état en fonction de l'état des entités voisines selon le voisinage du graphe.

Selon les situations de modélisation et les structures de graphes sur lesquelles on travaille, les opérateurs d'agrégation pourront être spécifiés soit dans la définition même d'une entité, soit dans la définition d'une fonction d'interaction. Le premier cas est plutôt adapté à traiter l'indétermination engendrée par les situations de partage de propriété. Le second permet de traiter les situations de multiplicité d'arcs incidents.

3.3.6 Agir sur la structure de Γ

On entend par opération sur la structure les opérations directement relatives au contenu de U d'un graphe d'interaction $\Gamma = (\Re, X, U)$.

Les analyses globales de graphe sont des fonctions qui portent sur l'ensemble d'un graphe, sans distinguer d'entité ou d'arc particulier. Les fonctions d'analyse locale de graphe font en revanche appel à certaines entités ou certains arcs choisis du graphe pour effectuer une mesure tenant compte des spécificités locales des éléments choisis.

Analyse globale de graphe

Dans les cas où un modèle comporte des graphes à structure dynamique, il est utile de disposer de fonctions permettant d'effectuer certaines mesures sur la structure elle même. Cela permet d'analyser comment cette structure évolue, pour savoir si certaines configurations sont apparues, ou si certaines contraintes sont conservées. Selon les résultats de ces mesures, le modèle pourra éventuellement orienter la simulation de sa propre évolution vers un scénario ou vers un autre. C'est là une possibilité qui est donnée au modélisateur

de construire un modèle ayant la capacité de s'auto-analyser en tant que système et influencer son propre comportement en fonction de la façon dont ce système évolue.

Les fonctions d'analyse qui portent sur un graphe dans sa totalité sont de la forme :

$$f : \Gamma \mapsto D_m \tag{3.36}$$

où Dm représente le domaine des valeurs résultant de cette analyse. On peut classer ces fonctions d'analyse selon le domaine D_m qui les concerne. A titre d'exemple on peut citer les fonctions de test et les fonctions de comptage :

Fonctions de test, de la forme $f : \Gamma \mapsto \{vrai, faux\}$ Ces fonctions peuvent par exemple servir à répondre à des questions comme :
- Γ est-il connexe ?
- Γ contient-il des entités isolées ?
- Γ est-il un arbre ?
- Γ est-il un graphe planaire ?

Fonctions de comptage, de la forme $f : \Gamma \mapsto N$ où N est l'ensemble des entiers naturels. De telles fonctions qui peuvent par exemple donner :
- Le nombre total d'entités de Γ : $card(X)$
- Le nombre total d'arcs de Γ : $card(U)$
- Le nombre d'entités isolées : $card(\{x \in X / d_\Gamma(x) = 0\})$
- Le nombre de composantes connexes de Γ
- Le nombre de points d'articulation du graphe

Les listes données ici à titre d'exemple sont bien évidemment loin d'être exhaustives. On peut imaginer bien d'autres fonctions d'analyse globale de graphe. En particulier les fonctions effectuant des mesures dont le résultat est un nombre réel. Sur certains graphes planaires topologiques, on peut effectuer des mesures relatives à la topologie qui peuvent s'avérer utiles dans les cas ou cette topologie est liée à une représentation spatiale modélisée sous forme de graphe.

Analyse locale de graphe

A partir d'éléments choisis d'un graphe, il est possible d'effectuer des analyses locales, en faisant usage de fonctions que nous classons ici selon les types d'éléments pris en considération :

Analyse à partir d'une entité On se donne un élément $x \in X$ et on mesure un certain nombre de choses à partir de ce x. Par exemple :
- Nombre d'arcs incidents $d^+_\Gamma(x)$ et $d^-_\Gamma(x)$
- Ensemble des entités voisines de x
- ...

Analyse à partir de plusieurs entités On se donne par exemple deux éléments $x \in X, y \in X$ et on mesure un certain nombre de choses entre ces x, y. Par exemple :
- Distance entre x et y dans le graphe

- Ensemble des voisins communs à x et de y
- ...

Analyse à partir d'un arc On se donne un arc $u \in U$ et on fait nos mesures à partir de ce u, comme par exemple :
- Entités à chaque extrémité de u
- Ensemble des arcs suivants ou précédents à u
- ...

Analyse à partir de plusieurs arcs On se donne une famille d'arcs $u_1, \ldots, u_k$ et on effectue des mesures sur cette famille. Par exemple :
- Test de connexité de l'ensemble
- Recherche de cycle
- Test de planarité
- ...

Là encore on ne peut donner une liste exhaustive des fonctions d'analyse locale qu'il est possible de construire et il faut laisser au modélisateur la possibilité de spécifier lui même les fonctions répondant à ses besoins. Tout comme les fonctions d'analyse globale, ces fonctions d'analyse locale peuvent être utilisées entre chaque changement d'état pour décider du scénario d'évolution que doit suivre le modèle.

3.3.7 Usage de plusieurs graphes d'interaction dans un même modèle

Soient $G = (\Re_G, X_G, U_G)$ et $H = (\Re_H, X_H, U_H)$ deux graphes d'interaction faisant partie d'un même modèle. Dans la situation où $X_G \cap X_H \neq \emptyset$ certaines entités présentes dans X_G sont aussi présentes dans X_H. On peut donc se trouver avec des entités x telles que

$$\exists x \in X_G \cap X_H, y_G \in X_G, y_H \in X_H, (x, y_G) \in U_G \wedge (x, y_H) \in U_H \tag{3.37}$$

Les entités faisant ainsi partie à la fois de G et de H sont des entités qui possèdent les propriétés pour jouer au moins un rôle défini dans $\Re_G$, mais aussi les propriétés pour jouer au moins un rôle défini dans $\Re_H$. On peut considérer que ce sont des entités qui participent au même modèle selon deux points de vue différents [46], et il est intéressant de pouvoir combiner des opérations selon chacun des points de vue.

3.3.8 Scénario

Les définitions que nous venons de donner, au niveau des individus (entités et rôles), et des interactions (relations et graphes d'interactions) sont déclaratives. Nous voulons par là signifier qu'il s'agit de descriptions de propriétés et de contraintes associées aux éléments d'un modèle, mais que l'ordre dans lequel ces opérations peuvent être effectuées lors d'une simulation n'est pas précisé. C'est au niveau du système que nous proposons d'indiquer l'enchainement des opérations, à travers le concept de scénario.

Un *Scénario* est une liste ordonnée d'opérations

$$S = (o_1, \ldots, o_n) \tag{3.38}$$

où les opérations o_i peuvent être :

- la construction d'entités d'après leur définition
- la construction de graphe d'interaction (d'après la définition d'une relation)
- l'appel d'une fonction de consultation ou de transition sur une instance d'entité
- l'appel d'une fonction de consultation, de transition, ou d'interaction sur un graphe d'interaction
- une fonction de contrôle de type test ou boucle répétitive

3.4 De la donnée géographique au graphe d'interaction

Nous avons fait le choix de définir et utiliser un concept de graphe d'interaction comme forme de représentation générique pour les éléments d'un modèle. Lorsque l'on souhaite construire un modèle de dynamique paysagère, la question se pose de la méthodologie de construction de tels graphes et des outils associés que l'on peut concevoir. On imagine aisément qu'il existe une infinie diversité dans la façon de construire des structures de graphes à partir de sources de données dont les formes sont bien souvent hétérogènes. Y a-t-il un risque que l'on doive élaborer une méthodologie de construction de graphe différente à chaque nouveau modèle que l'on souhaite construire ? Est-il possible, malgré cette diversité, de proposer un nombre limité d'outils qui aident les modélisateurs à construire ces graphes, sans pour autant contraindre leur capacité d'expression ?

Nous avons tenté de répondre à ces questions en deux temps. Dans cette section d'abord, en cherchant les facteurs qui nous permettent de constituer des catégories dans cette diversité, c'est à dire dans les formes de sources de données, dans les méthodes de construction, et dans les formes de structures de graphes auxquelles on peut aboutir. Dans le chapitre suivant ensuite (voir 4.6), en proposant un outil à travers un élément de langage (nommé *datafacer*) qui permettra d'implémenter des liens entre sources de données et construction de graphes d'interaction.

3.4.1 Sources de données

Les formes de sources de données les plus courantes auxquelles on peut faire appel pour modéliser des paysages sont les couches vectorielles de SIG, les images de télédétection, et des tableaux de mesures effectuées sur le terrain.

Les couches vectorielles de SIG sont aujourd'hui très largement représentées sous la

forme de tables, au sens des bases de données relationnelles[9]. Il est en effet possible de stocker de l'information géographique sous la forme de tables, dont chaque ligne est un n-uplet de valeurs prises dans différents domaines, l'un au moins de ces domaines étant de nature géométrique (point, ligne ou polygone par exemple). Ces tables peuvent être manipulées par les opérations habituelles de l'algèbre relationnelle (projection, sélection, jointure, etc.). L'adjonction d'un module d'opérateurs spatiaux à un système de gestion de base de données (SGBD) permet en outre de faire appel à des fonctions spécifiquement spatiales portant sur les attributs géométriques de ces tables [130]. Cette façon de stocker l'information géographique dans des bases de données à extension spatiale a fait l'objet d'un processus de standardisation par l'Open Geospatial Consortium [8] qu'une grande majorité de logiciels SIG a aujourd'hui adopté.

Les formats de fichiers utilisés par des logiciels SIG communs comme ArcGIS ou MapInfo correspondent eux aussi à des n-uplets dont l'un des domaines de valeurs est de type géométrique. Ces formats reproduisent en fait l'équivalent des tables de SGBD-spatiaux mais sous la forme de fichiers. On peut donc dans le principe considérer ces différentes formes de représentations comme des tables au sens des bases de données relationnelles, que celles-ci soient stockées dans une base de données ou dans des fichiers. Dans la pratique une différence d'importance existe cependant dans la façon d'accéder à ces différentes sources de données : il faut faire appel à des bibliothèques logicielles spécifiques pour accéder à chacun des formats de fichier, ou au langage SQL pour accéder aux tables d'une base de données. Selon le format de stockage, on aura donc accès à des ensembles de fonctionnalités plus ou moins riches pour manipuler ces sources de données : sur une base de données dotée d'une extension spatiale, on dispose de nombreuses fonctions pour agir sur les attributs thématiques et géometrique et d'un langage (SQL) pour les combiner alors sur des fichiers on ne peut que lire le contenu tel quel.

Les images de télédétection peuvent difficilement être utilisées telles quelles sans traitement dans un contexte de modélisation. Nous ne parlons pas ici des pré-traitements éventuels d'orthorectification ou d'étalonnage de la radiométrie, qui sont plutôt du ressort des fournisseurs d'images. Nous parlons de séries d'opérations de nature à réduire la quantité d'information et à l'organiser. Ce traitements doivent faire partie d'une inévitable étape de préparation des données en vue de leur intégration dans un modèle. Il s'agit par exemple d'opérations de segmentation d'images, ou de différentes formes de classifications. Ces types de traitements aboutissent le plus souvent soit à des données de type vectorielles destinées à être intégrées dans un SIG (donc des tables), soit à une image dont les pixels ont été regroupés en un nombre fini de classes.

On peut dans ce dernier cas faire appel à des opérations que l'on appelle des fonctions zonales pour extraire de l'image des mesures statistiques localisées. Ces mesures peuvent alors prendre la forme de tables pour être intégrées dans un SIG. On peut aussi imaginer conserver l'image et la considérer comme un graphe dont les sommets sont les pixels et les arcs sont définis de manière implicite par un voisinage similaire à ceux que l'on utilise dans les automates cellulaires.

[9]Nous pourrions utiliser le terme *Relation* à propos de ces tables mais nous voudrions éviter qu'il y ait confusion avec l'usage spécifique que nous faisons de ce terme à plusieurs reprises dans ce document

Remarquons que ce l'on vient de décrire sur les images de télédétection peut s'appliquer aux modèles numériques de terrains. On fait appel à ce type de données lorsque l'on a besoin d'intégrer des informations issues du relief dans un modèle. On effectuera dans ce cas des opérations de préparation analogue à celles que nous avons évoquées pour les images de télédétection mais les opérateurs servent plutôt à extraire des indications d'altitude, de pente, d'orientation du terrain, ou d'écoulement des eaux.

Les mesures de terrain sont souvent géoréférencées, soit de manière directe en enregistrant par exemple une position à l'aide d'un GPS[10], soit de manière indirecte à l'aide d'une adresse, ou d'une position dans une grille d'échantillonnage. Ces données se présentent comme des n-uplet de valeurs dont certaines représentent les coordonnées (ou une adresse) du lieu de la mesure. Cela revient là aussi à une forme de table semblable à celles que nous venons de décrire pour les couches de SIG, seul le format du fichier de stockage est différent.

On constate que sur le fond, les principales catégories de sources de données auxquelles on peut être confronté sont peu nombreuses. La diversité relève davantage de questions de forme, en particulier lorsqu'il s'agit de fichiers dont les formats sont propres aux logiciels qui les produisent.

Il y a bien entendu aussi une diversité du contenu qu'il nous faut examiner, et qu'il faut croiser avec l'intention du modélisateur. C'est là l'objet de la section qui suit.

3.4.2 Construction de graphes

Nous avons évoqué en 1.3 la nécessité du choix d'un point de vue dans la définition, la description, ou la modélisation d'un paysage. Dans le cas de la modélisation, ce choix dépendra directement de l'intention du modélisateur : des phénomènes qu'il désire étudier, de la vision propre à son domaine d'expertise, mais aussi des niveaux d'échelle qu'il choisira pour traiter les données et construire son modèle. Pour concevoir des outils d'aide à la construction de graphes d'interaction, nous avons cherché à nous faire une idée de la diversité des situations dans lesquelles les modélisateurs peuvent se trouver. Nous avons abordé cette question en recherchant dans la littérature des expériences d'analyse ou de modélisation de paysages où il est fait usage de graphes. Nous avons étudié en particulier les méthodologies de construction de graphes mises en oeuvre dans ces expériences.

La majorité des expériences de ce type dont nous avons eu connaissance ont été menées dans le domaine de l'écologie du paysage. Il s'agit souvent d'études de diffusion de populations d'organismes vivants de toutes sortes, et de l'influence de la fragmentation et de la connectivité de leurs zones d'habitat sur cette diffusion.

M.D. Cantwell et R.T.T. Forman ont proposé en 1993 [31] une méthode de construction de graphes à partir de photo-interprétation d'images de télédétection. Ils ont basé leur analyse du paysage sur deux hypothèses :

[10] Global Positionning System

1. Le paysage est constitué de taches ou de zones homogènes, que l'on choisit de distinguer de leur environnement. Ces taches constituent les éléments du paysage et peuvent être séparées par des frontières, ou reliées par des corridors [56], qui eux aussi sont considérés commes des éléments du paysage. Dans le cas présent, les auteurs ont choisi d'intégrer l'environnement de ces taches comme étant lui même un élément du paysage, ce qui permet une prise en compte de la totalité de l'espace occupé par ce paysage.
2. Les éléments d'un paysage spatialement adjacents sont susceptibles d'interagir les uns avec les autres, même s'ils sont de natures différentes.

Les graphes ont été construits en prenant pour sommet chaque élément du paysage, et en définissant une arête pour chaque adjacence spatiale entre deux éléments. Les éléments de type corridor sont susceptibles d'être reliés les uns aux autres d'une manière à la fois spatiale et fonctionnelle même si l'on n'observe pas de limite entre eux sur les photos aériennes. C'est par exemple le cas de croisement de routes, ou de bras de rivière qui se rejoignent. Une arête est construite à chaque adjacence de ce type entre deux corridors (figure 3.13).

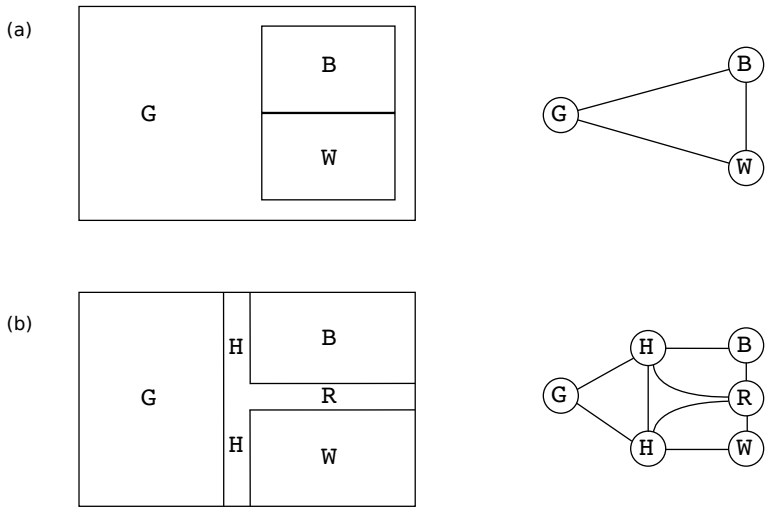


Fig. 3.13 – Eléments de paysage et leur graphes correspondants. G (grassland) représente une prairie, B (bean field) représente un champ de haricots, W (woods) représente une zone boisée. En (a) on voit que B et W sont spatialement inclus dans G, et que B et W sont adjacents. Les adjacences spatiales que cela représente sont traduites par des arêtes sur le graphe de droite. En (b) on a une route principale H (highway) et une route secondaire R (road) qui séparent G, B et W. On observe aussi que R rejoint H par un croisement en forme de T. Les différentes adjacences que cela représente sont traduites sur le graphe de droite. (D'après Cantwell et Forman, 1993 [31])

Il est intéressant de noter qu'en effectuant ce travail systématique sur 25 paysages différents, ils ont pu identifier un nombre relativement restreint de motifs que l'on retrouve dans les graphes obtenus. Ces motifs sont en quelque sorte représentatifs de ce que l'on pourrait considérer comme des briques élémentaires de la constitution de paysages (figure

3.14). Chaque modèle de paysage a produit un graphe qui porte une combinaison unique de ces briques. Mais certains de ces motifs ont été observés fréquemment : ceux qui sont nommés `Necklace`, `Spider` et `Graph cell` dans la figure 3.14 ont par exemple été détectés dans plus de 90% des graphes.

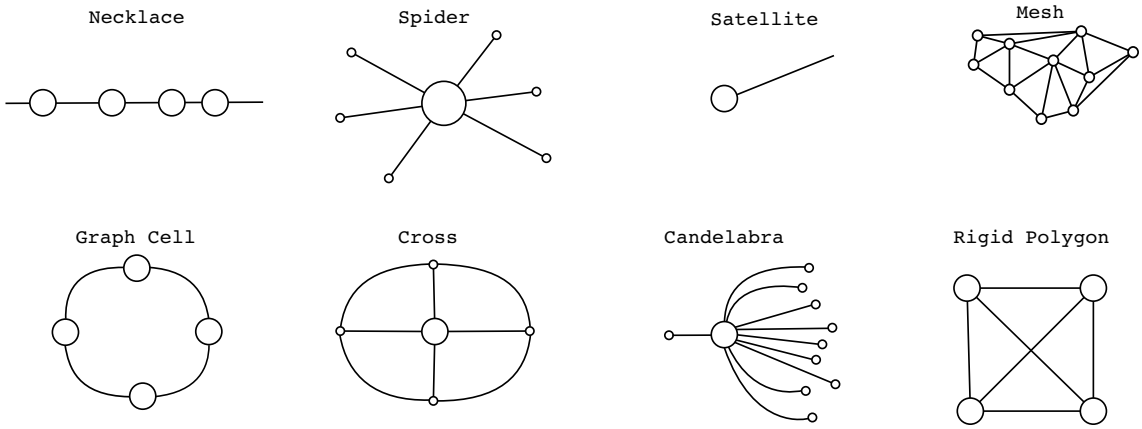


FIG. 3.14 – Exemples de motifs observés dans les structures de graphes produits par l'analyse de 25 paysages différents. Les sept premiers (de gauche à droite et de haut en bas) étant les plus communs. (D'après Cantwell et Forman, 1993 [31])

T.H. Keitt et al. [88] ont poursuivi ce travail de recherche méthodologique sur la construction de graphes en mettant l'accent sur deux aspects :

1. La méthode basée sur la photo-interprétation de Cantwell et Forman présentant un caractère subjectif, Keitt et al. ont cherché à adopter une démarche plus objective dans la détection des éléments paysagers et des relations d'adjacence entre ces éléments. Les images subissent un traitement automatisé qui est une forme de classification destinée à séparer les pixels qui représentent un habitat pour une espèce donnée des autres pixels qui sont alors classés comme environnement. Chaque groupe de pixels adjacents d'habitat constitue alors un élément du paysage qui va être représenté par un sommet dans le graphe.
2. Les résultats de l'analyse de paysages étant très dépendants de l'échelle à laquelle on se place, ils ont mis au point une méthodologie d'analyse multi-échelles. Plutôt que de s'appuyer sur une adjacence spatiale directe, ils se sont basés sur la distance minimale entre deux zones d'habitat : si cette distance est inférieure à un seuil donné, une arête est ajoutée au graphe qui relie les sommets correspondants. Le calcul de l'adjacence est donc paramétrable par le choix de ce seuil. En faisant varier ce seuil, ils ont obtenu différents graphes sur lesquels ont été appliquées des mesures d'indices de connectivité issus de travaux sur la théorie de la percolation.

Ces travaux ont été complétés quelques années plus tard par une étude de l'effet de deux types de changements sur la connectivité d'un graphe de connectivité d'habitats : la supression d'arêtes et la supression de sommets [140]. Les résultats montrent que l'on

peut tirer de ces études de sensibilité de riches enseignements sur l'importance relative de certains sommets sur la connectivité de l'ensemble. Enfin les auteurs de cette étude ont expérimenté la construction de l'arbre couvrant minimum qui présente un intérêt au sens écologique en tant que squelette de la connectivité de l'habitat d'une espèce sur un territoire.

En 2007 A. Fall et al. se sont inspiré des travaux que nous venons d'exposer, et des travaux de A.Okabe et al. ([118]) sur les différentes formes de construction de diagrammes de Voronoi, pour développer le concept qu'il nomment "spatial graph theory" [51].

Il s'agit de ne pas détacher complètement un graphe du référentiel géographique et des structures spatiales qu'il représente. Ainsi, au lieu de concevoir des sommets sans dimension, les sommets d'un *graphe spatial* sont des objets à deux dimensions qui ont une position, une surface et une forme. Les arêtes relient deux sommets en des points géoréférencés qui peuvent se trouver soit sur le périmètre des formes associées à ces sommets, soit sur leurs centroides, selon les besoins du modèle.

Un graphe spatial est défini formellement de la façon suivante :

$G = (N, L, S, f_{cost})$ est un graphe pondéré (N, L), combiné à un domaine spatial S et une fonction spatiale de coût f_{cost} dans lesquels :

- les sommets de l'ensemble N représentent des zones de S, contiguës et distinctes ;
- chaque arête $l = ((n, m), (pn, ..., pm))$; $l \in L$; n,$m \in N$; représente une connexion entre n et m, ainsi qu'une ligne entre les points pn et pm qui sont, respectivement, dans les zones des sommets n et m ;
- le poids d'une arête l est défini par le cumul des coûts tout le long du trajet dans S qu'elle repésente.

Les auteurs ont en outre complété ces définitions par une généralisation de la triangulation de Delaunay qu'il ont nommé *graphe planaire minimal*[11] (figure 3.15). Ce graphe planaire minimal (GPM) possède des propriétés intéressantes : il peut être considéré comme une approximation du graphe complet, et l'arbre couvrant minimal est un sous-graphe du GPM.

Le fait de conserver la forme et le géoréférencement des sommets permet de garder un lien avec des sources de données de type classification d'image de télédétection ou modèle numérique de terrain pour les utiliser dans des calculs de coût de trajet par exemple. Ces graphes spatiaux sont une synthèse et une formalisation des différentes constructions de graphes que nous avions vues précédemment. Ils permettent en particulier de décrire formellement des graphes issus de différentes méthodes de construction des arêtes : voisinages basés sur une distance minimale (entre les centroides ou les bords des zones d'habitats), voisinage basé sur le coût d'un trajet ou encore adjacence de zones d'influence (ces zones sont dans ce cas définies par le Diagramme de Voronoi dual du GPM).

On trouve de nombreux travaux dans la littérature dans lesquels les différentes métho-

[11] version originale : *minimum planar graph*.

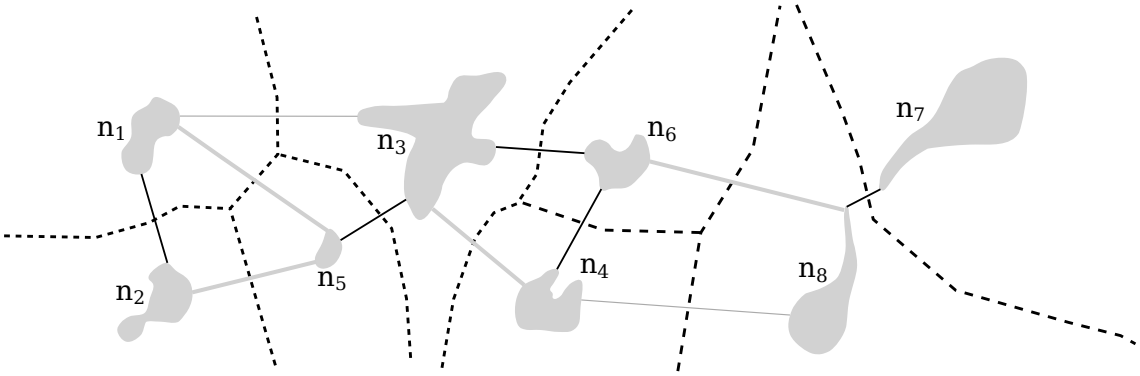


FIG. 3.15 – Graphe spatial comportant les sommets n1,...,n8 (zones grises). Les arêtes sont représentées par des lignes reliant les sommets. L'ensemble des arêtes constituent le graphe planaire minimal. Les lignes noires constituent le sous-graphe des plus proches voisins, les lignes grises fines représentent les arêtes supplémentaires pour obtenir l'arbre couvrant minimal et les lignes grises épaisses complètent le GPM. Les lignes pointillées représentent les frontières du diagramme de Voronoi dual du GPM. (D'après Fall et al., 2007 [51])

dologies que nous avons présentées ont été mises en oeuvre. Dans les quelques exemples que nous citons ci-dessous, les modélisateurs ont à chaque fois choisi des entités géoréférencées comme sommets de leurs graphes. Ils ont ensuite fait appel à des modèles que l'on peut souvent assimiler au calcul d'un coût de trajet dans un environnement entre deux entités pour décider si ces entités sont reliées ou non par une arête (ou un arc si le graphe est orienté) :

- Pour étudier l'impact de la construction de barrages sur des populations de saumons, R. S. Schick et S. T. Lindley[134] ont choisi de construire des graphes dans lesquels chaque sommet représente une population de saumon. Chaque population est liée à la rivière dans laquelle les individus sont nés et retournent se reproduire. Les sommets du graphe sont géoréférencés avec une position qui correspond au point d'intersection entre la rivière de la population concernée et la courbe de niveau de 500m d'altitude. Le calcul de construction des arcs est fonction de la distance "à la nage" le long de bras de rivière.
- Lors de travaux portant sur les populations de corail dans les zones tropicales de l'océan pacifique [137], les sommets du graphe correspondent à des récifs coraliens avec leur position géographique. Le arcs sont établis à partir d'une estimation de la proportion de larves de corail susceptibles de passer d'un récif à un autre en étant simplement transportées par les mouvements de l'océan. Cette estimation est calculée à l'aide d'un modèle hydrodynamique existant par ailleurs.
- Ce sont des unités administratives qui ont été choisies par M.L. Margosian et al. comme sommets pour la construction d'un graphe dédié à l'analyse de risque de diffusion de pathogènes sur des cultures [98]. Chaque sommet est géographiquement positionné sur le centroide de l'unité administrative qu'il représente. Les arêtes représentent simplement un voisinage d'adjacence spatiale entre les unités adminis-

tratives. Un indice de résistance à la transmission de pathogènes est calculé à partir de données locales et environnementales, et cet indice est attaché à chaque arête.
- Pour l'étude de certaines dynamiques urbaines [119], D. O'Sullivan utilise les batiments, et parfois les rues, avec leur position et une forme simplifiée pour les sommets du graphe qu'il construit. Ils propose différentes solutions pour établir des arêtes : une triangulation de Delaunay reliant les centroides des batiments, les distances entre centroides inférieures à un seuil, la visibilité mutuelle entre les centroides des batiments ou encore la visibilité mutuelle des portes d'entrées des batiments bordant les rues.

A l'exception de la thèse de D. O'Sullivan sur les dynamiques urbaines que nous avons cité en dernier, il faut reconnaître que ces travaux reflètent surtout les besoins en écologie du paysage. Il s'agit en outre d'une utilisation de graphes à des fins d'analyse de connectivité et non de simulation de dynamiques paysagères. C'est la raison pour laquelle nous ne rentrons pas dans les détails de l'usage de ces graphes en écologie. Le lecteur intéressé par ces aspects pourra consulter deux articles de synthèse : celui de B. Rayfield et al. [128] qui traite particulièrement des fonctions de voisinages basées sur un calcul de coût de trajet entre zones d'habitats ; et celui de D.L. Urban et al. [141] qui traite d'une manière plus générale de la façon dont ces modèles de connectivité peuvent être construits, paramétrés, et testés, dans un contexte d'analyse et de conservation des espèces.

Pour guider nos choix dans la construction d'un outil, nous retiendrons les points suivants :
- La transformation de différentes sources de données en entités qui pourront constituer les sommets d'un graphe présente deux aspects : l'un strictement technique, est relatif aux formats des fichiers ou des bases de données auxquels on doit accéder ; les outils que l'on peut fournir pour cela peuvent rester indépendants de tout modèle. L'autre dépend davantage de la façon dont le modélisateur souhaite traiter les données pour en extraire des entités. Il est donc important de lui laisser toute liberté pour décrire ces traitements.
- Nous avons vu qu'il est souvent intéressant de disposer d'abord d'un graphe complet (ou d'une approximation comme un GPM) dont on va ensuite extraire un sous-graphe en ne retenant que les arcs qui répondent à des critères que l'on peut paramétrer.
- Il peut être utile d'intégrer au graphe des informations spatiales comme la position ou la forme des entités, ou encore le chemin correspondant à une arête dans un environnement géographique.

3.5 Conclusion

Nous avons proposé dans ce chapitre les définitions formelles d'un ensemble de concepts dédiés à la modélisation et la simulation de dynamiques paysagères. Nous avons fait le choix d'une forme de modélisation basée sur la construction et l'usage de graphes. Ces graphes représentent les différentes interactions qui peuvent intervenir dans un modèle, et nous avons défini leur nature et leur usage selon trois niveaux :

Individus Les sommets de graphes sont constitués d'individus qui sont formalisés par le concept d'*entité*. Ces entités possèdent un état sous la forme de *propriétés*. Si nécessaire les états passés des entités peuvent être conservés. Des fonctions de *consultation* et de *transition* permettent respectivement de lire l'état des entités, et de le modifier.

Intéractions Nous avons formalisé la façon dont des graphes peuvent porter la sémantique associée à des interactions entre catégories d'entités. Ces catégories sont définies à travers le concept de *rôle*, et le concept de *relation* regroupe l'ensemble des fonctions pouvant agir sur des rôles donnés. A partir d'une définition de relation, on peut produire un ou plusieurs *graphes d'interaction*. Des fonctions définies dans les relations peuvent être appliquées sur ces graphes. Les fonctions d'interaction en particulier permettent d'agir simultanément sur l'ensemble d'un graphe. La combinaison de cet aspect simultané des interactions et de certaines topologies de graphes peut parfois poser problème. Nous avons en effet identifié certaines situations dans lesquelles les changements d'état de propriétés pouvaient être indéterminés, et pour éviter d'avoir à poser des contraintes trop fortes sur la construction des graphes, nous avons préféré définir des *fonctions d'agrégation* qui permettent de traiter ces cas d'indéterminations.

Système La construction de graphes d'interaction, et l'enchaînement des transitions d'états que l'on fait subir à ces graphes, sont spécifiés dans un *scénario*. Un scénario peut ainsi manipuler et synchroniser l'évolution de plusieurs graphes d'interaction, chaque graphe représentant un aspect particulier (ou un point de vue) du système que l'on modélise.

Le choix de structures de graphes comme primitives de modélisation a été principalement motivé par deux objectifs : (1) être capable de représenter des intéractions de toutes natures, y compris spatiales, avec un même formalisme commun ; et (2) focaliser l'exercice de modélisation sur un niveau intermédiaire entre l'individu et le système.

Ce choix comporte cependant des risques, en particulier celui de rendre fastidieux la construction de ces graphes. Nous avons étudié les questions liées au passage de différentes formes de données aux graphes d'interactions. Nous avons dégagé de ce travail quelques points importants qui vont guider la mise au point d'outils permettant d'aider à l'automatisation de la construction de graphes.

La définition de ces concepts est un préalable à la construction d'un outil de simulation. Nous verrons dans le chapitre suivant que l'on retrouve ces concepts sous la forme d'éléments d'un langage de modélisation.

Chapitre 4

Ocelet

Sommaire

4.1 Introduction

Ce chapitre est consacré à Ocelet, le langage métier que nous avons développé pour aider à la modélisation spatiale et temporelle à des fins de simulation [45]. La conception d'Ocelet correspond au deuxième sous-objectif que nous nous sommes donné : proposer un outil de simulation basé sur les concepts exposés dans le chapitre 3. Cet outil devant permettre la construction d'un état initial, la description de règles qui régissent la façon dont le système peut évoluer, et l'exécution d'expériences de simulation.

Ce langage est construit autour des trois niveaux exposés dans le chapitre précédent : individus, interactions et dynamique du système, à travers les concepts d'*Entité* présenté en section 4.3, *Relation* présenté en section 4.4 et *Scenario* présenté en section 4.5. Le concept d'entité permet de décrire les éléments d'un modèle susceptibles d'interagir les uns avec les autres. Le concept de relation représente la nature des interactions et leur instanciation permet de construire des graphes d'interaction qui portent la structure. Enfin le concept de scénario permet de décrire la dynamique d'un modèle, c'est à dire construire un état initial, puis indiquer l'ordre dans lequel les interactions et les opérations sur les structures de graphes doivent avoir lieu.

A ces trois concepts de base, s'ajoutent un certain nombre d'éléments et propriétés du langage que nous avons ajoutés pour faciliter l'écriture de modèles. Ce sont les *datafacer*, *group*, *list*, *structure*, *variables historiques* qui sont présentés en section 4.7.

Le choix de donner à nos travaux la forme d'un langage métier est le résultat d'un compromis : nous nous trouvons à mi-chemin entre la bibliothèque de fonctions pour un langage généraliste, et une plateforme intégrée de modélisation regroupant par exemple des outils de modélisation graphique ou des modèles pré-définis que l'on peut paramétrer.

Avec une bibliothèque de fonctions ou API[12], on peut apporter de nouvelles structures de données et de nouvelles fonctions à un langage de programmation existant. Cela revient en quelque sorte à étendre les possibilités d'un langage (souvent un langage généraliste comme C++, Java, Lisp, etc.) en proposant des modules pré-programmés que l'on peut assembler et paramétrer tout en bénéficiant de la richesse d'expressivité de ce langage. Cette forme d'extension offre un gain de temps important : on n'a pas besoin de programmer soi-même un certain nombre de fonctions, ni d'acquérir l'expertise nécessaire pour les programmer. C'est aussi potentiellement un gain en qualité puisque ces bibliothèques parfois complexes doivent avoir fait l'objet de tests et doivent en principe respecter des spécifications précises et documentées.

Une API est une forme d'extension qui n'affecte en rien la sémantique de base du langage auquel elle est dédiée. Par exemple une API développée pour Java ou C++ apportera de nouvelles classes mais restera fondamentalement orientée objet, et une bibliothèque pour Lisp apportera de nouvelles fonctions mais restera fondamentalement basée sur la

[12] API : "Application Programming Interface", c'est à dire un certain nombre de définitions, de structures de données et de fonctions permettant de les manipuler, sous la forme d'une bibliothèque.

programmation fonctionnelle.

Proposer un langage métier c'est proposer une nouvelle forme d'expression, et cela se justifie particulièrement si cette forme d'expression reflète et accompagne une nouvelle façon de penser lors de la conception d'un programme. La conception de modèle de dynamiques paysagères faisant appel à un ou plusieurs graphes d'interactions nécessite une réflexion particulière sur le système que l'on cherche à modéliser : il faut par exemple réfléchir à la topologie des graphes d'interaction que l'on va construire et aux caractéristiques de ces graphes, il faut aussi imaginer les processus qui vont être mis en œuvre dans les relations et qui vont produire des évolutions simultanées sur les groupes d'entités interconnectées dans ces graphes. Cette façon de concevoir des dynamiques spatiales reflète une sémantique particulière qu'un langage métier est mieux à même d'intégrer qu'une bibliothèque de fonctions de type API.

On peut analyser notre démarche de construction d'Ocelet à travers les cinq phases qu'a proposées M. Mernik [102] à propos du développement d'un langage métier : décision, analyse, conception, implémentation et déploiement.

Décision Les arguments qui nous ont amenés à choisir de construire un langage métier plutôt qu'une autre forme d'outil de modélisation sont exposés ci-dessus.

Analyse La phase d'analyse a abouti à la définition d'un certain nombre de concepts qui sont détaillés dans le chapitre 3 : entité, rôle, relation, graphe d'interaction et à la description de la façon dont les graphes d'interaction peuvent évoluer dans le temps.

Conception Les syntaxes abstraite et concrète du langage Ocelet ont été définies dans la phase de conception. Cela a permis d'écrire la grammaire du langage, préparant le travail pour construire un compilateur. Les éléments du langage sont présentés dans les sections suivantes du présent chapitre.

Implémentation Cette partie requiert un important travail d'ingénierie en développement de logiciel. Deux modules principaux ont dû être développés : le compilateur d'une part, et le moteur d'exécution d'autre part. Le rôle du compilateur est de traduire les programmes écrit en Ocelet dans un langage de programmation généraliste que l'on appelle le langage cible (dans l'implémentation que nous avons produite, le langage cible est Java). Le moteur d'exécution est une bibliothèque de fonctions directement développées dans le langage cible. Le code généré par le compilateur s'appuie sur des appels aux fonctions du moteur d'exécution. Ce moteur intègre en particulier la sémantique relative à la dynamique des graphes d'interaction et la gestion de la simultanéité de changements d'état des entités présentes dans ces graphes. Cette phase d'implémentation fait l'objet de la première partie du chapitre suivant relatif à l'outillage d'Ocelet.

Déploiement Afin de faciliter l'adoption d'Ocelet et son utilisation, nous avons entrepris la construction d'un environnement de développement basé sur *Eclipse Rich Client Platform*[2]. Cet environnement intègre un éditeur, le compilateur, le moteur d'exécution, le regroupement des éléments d'un modèle sous la forme de projet, et quelques outils annexes pour exploiter les résultats de simulations. Le travail réalisé est décrit dans la deuxième partie du chapitre suivant (5.4).

Nous présentons dans les sections suivantes les différents éléments d'Ocelet en commençant par les concepts les plus importants spécifiés dans le chapitre 3 : modèle, entité, relation, scenario. C'est ensuite le lien entre diverses formes de sources de données et les graphes d'interaction qui est exposé à travers le concept de Datafacer. Les autres éléments du langage (types de base, collections, instructions de contrôle et quelques autres fonctions) sont présentés en dernier. Cet ordre correspond plus ou moins à l'ordre dans lequel se déroule la construction d'un modèle : on cherche d'abord à se faire une idée globale du modèle, en particulier quelles sont les entités, qui va interagir avec qui, de quelle façon et dans quel ordre. On s'intéresse ensuite à la construction de ces éléments à partir de données dont on dispose. Enfin on s'occupe des détails de la programmation elle même.

4.2 Modèle

Construire un modèle à l'aide d'Ocelet, revient à écrire un programme dont l'exécution permettra de simuler l'évolution des variables de ce modèle. Un tel programme est constitué de différents éléments : des définitions d'entités qui vont être utilisées dans le modèle, des définitions de relations à partir desquelles on pourra construire des graphes d'interaction, et au moins un scénario décrivant les séquences d'opérations à effectuer lors des simulations. A cela peuvent s'ajouter quelques éléments complémentaires, comme par exemple l'importation de morceaux de modèles stockés dans d'autres fichiers, ou la déclaration d'utilisation de datafacers (voir 4.6) qui sont utilisés dans le modèles.

Un modèle Ocelet est un programme constitué de la façon suivante[13] :

```
['uses' <fichier ocelet>]*
['datafacer' <nom d'un datafacer>]*
['structure' <définition d'un type composite>]*
['entity' <définition d'entité>]*
['relation' <définition de relation>]*
['affect' <définition d'opérateur d'agrégation>]*
('scenario' <définition de scénario>)+
```

Comme un modèle peut contenir plusieurs scénarios, le moteur d'exécution a besoin de connaitre celui qui sert de point d'entrée, c'est à dire celui qui doit être exécuté en premier pour effectuer une simulation. Nous avons décidé de donner au scenario servant de point d'entrée le même nom que le nom de fichier du modèle lui-même (sans l'extension). Ainsi, si un modèle se nomme `Mangrove.oclt`, il doit contenir au moins un scenario déclaré par : `scenario Mangrove {...}` qui sera le point d'entrée pour exécuter une simulation.

[13] Les notations utilisées pour décrire la syntaxe du langage sont indiquée dans la partie *Notations* en fin de document.

4.3 Entité

Les entités d'Ocelet correspondent aux éléments d'un modèle tels que nous les avons définis en 3.3.1 : les entités possèdent un état constitué d'un ensemble de propriétés, et elles peuvent interagir les unes avec les autres lorsqu'elles sont reliées à travers les arcs d'un graphe d'interaction.

En Ocelet, on doit définir chaque type d'entité que l'on veut utiliser dans un modèle. Une fois qu'un type d'entité a été défini, il est possible de créer un ou plusieurs exemplaires d'entités correspondant à cette définition. Le mécanisme est ici comparable à ce que l'on trouve dans les langages orientés objet : on définit une classe, puis on crée des instances de cette classe qui sont autant d'objets que l'on peut manipuler.

4.3.1 Définition d'un type d'entité

Un type d'entité est défini par une expression de la forme suivante :

```
'entity' <identifiant> '{'
  [définition de propriété]*
  [définition de service]*
'}'
```

Le code interne d'une entité peut contenir un ensemble de déclarations de propriétés et de déclarations de services. Les propriétés sont les éléments qui permettent de constituer l'état d'une entité. Les services sont des fonctions qui peuvent agir sur les propriétés.

Voici par exemple une définition valide d'entité :

```
entity Station {
  property real temperature;
  property int altitude;

  service real tempCelsius() {
    return temperature;
  }
  service real tempFahrenheit() {
    return (9/5)*temperature+32;
  }
}
```

4.3.2 Propriété

Déclaration

Chaque propriété doit être déclarée avec le mot clé *property* en spécifiant au minimum un type et un identifiant :

```
'property' <type> <identifiant> ['=' <expression d'initialisation>]';'
```

Les types de propriétés autorisés dans ces déclarations sont limités aux types de base du langage, et aux types composites (détaillés en 4.7.2).

Exemple de quelques déclaration valides :

```
property real temperature;
property text locationName;
property Poslatlon pos;
```

Les deux premières déclarations de cet exemple font usage de types de base d'Ocelet. Le troisième exemple fait appel à un type composite (que nous nommons `structure`), c'est à dire un type déclaré ailleurs dans le modèle et regroupant un ensemble de types de base sous un même identifiant. Les structures sont décrites plus en détail dans la section 4.7.2.

L'initialisation d'une propriété est possible lors de la déclaration avec deux formes possibles selon que l'on utilise un type de base ou un type composite :

```
property real temperature = 10.5;
property text locationName = "Myvatn";
property Poslatlon pos = Position{lat = 65.36; lon = 17.0; };
```

Propriétés historiques

Comme nous l'avons défini en 3.3.1, dans certains modèles on peut avoir besoin de conserver un historique des valeurs prises par une propriété pour effectuer par exemple des calculs qui doivent prendre en compte les états passés. Nous avons doté Ocelet d'une capacité à conserver l'historique discret des variables : on conserve les valeurs d'une propriété pour un nombre fini de valeurs passées. La syntaxe utilisée pour déclarer une propriété historique de ce type est la suivante :

```
'property' <type> <identifiant>@<profondeur>';'
```

La syntaxe est similaire pour accéder à l'historique d'une propriété :

`<propriété>@<index>` ; donne la valeur de la propriété à l'index historique spécifié.

Par exemple on déclare ici une propriété de type `real` d'une profondeur d'historique de 5 états passés. Chaque nouvelle valeur affectée à la propriété a pour effet de repousser en quelque sorte tout l'historique d'un pas vers le passé.

```
property real temperature@5;

 ...

temperature = 12.4;   // temperature@0 vaut 12.4
temperature = 12.6;   // temperature@0 vaut 12.6 et temperature@
   -1 vaut 12.4
```

Pour obtenir respectivement la valeur actuelle, puis la valeur précédente, puis 3 pas en arrière, on écrira :

```
t = temperature;
tt = temperature@-1;
ttt = temperature@-3;
```

On notera que `temperature` et `temperature@0` sont deux notations équivalentes pour obtenir l'état actuel d'une propriété.

4.3.3 Service d'entité

Les services d'une entité sont des définitions de fonction, c'est à dire un bloc de programme doté d'un identifiant, qui peut prendre des arguments en entrée et renvoyer éventuellement une valeur en retour. Ces services correspondent aux concepts de fonction de consultation et de transition définis en 3.3.1, il peuvent donc servir soit à interroger l'entité sur son état, soit à modifier cet état.

La définition d'une entité peut comporter plusieurs définitions de services. La définition d'un service est semblable à la définition de fonctions ou de méthodes dans d'autres langages. Elle prend la forme :

```
'service' [type] <identifiant>'('[liste d'arguments]')' '{'<code du service>'}'
```

Le type de retour n'est nécessaire que si le service renvoie une valeur. Dans cet exemple le premier service renvoie une valeur, et le second ne renvoie rien :

```
property real tempCelcius;

 ...

service real tempFahrenheit() { return (9/5)*tempCelsius+32; }

service setTempFahrenheit(real tf) { tempCelsius = (tf-32)*(5/9);
    }
```

Un service peut faire appel à un autre à l'intérieur d'une même entité. Par contre, une entité ne pouvant avoir une référence directe vers une autre (on ne peut pas avoir une propriété ayant pour type une entité par exemple), il n'est pas possible de faire un appel direct à un service d'une autre entité. Seuls les services de relation et les scénarios peuvent faire appel aux services d'une entité depuis l'extérieur de cette entité.

4.3.4 Instanciation, initialisation et utilisation

La définition d'une entité sert de modèle pour instancier un ou plusieurs exemplaires de cette entité. Il est possible d'initialiser des propriétés au moment de l'instanciation. La création d'une instance est une expression de la forme :

```
[type d'entité] <identifiant>'='<type d'entité>'{'(<propriété>=<valeur>';')*'}'';'
```

Prenons a titre d'exemple une entité définie par :

```
entity Station{
  property real temperature;
  property int altitude;

  service real tempFahrenheit() { return (9/5)*tempCelsius+32; }

  service setTempFahrenheit(real tf) { tempCelsius = (tf-32)*(5/9)
     ; }
}
```

Toutes les formes d'instanciation utilisées dans l'exemple suivant sont valides :

```
Station s1 = Station{};
s2 = Station{};
s3 = Station{temperature=2.3;};
s3 = Station{altitude=1453; temperature=2.3;};
```

L'accès aux propriétés d'une instance d'entité est direct, en écriture à travers une expression de la forme :

```
<entité>'.'<propriété>'='<valeur>;
```

Et en lecture à travers une expression de la forme :

```
<variable>'='<entité>'.'<propriété>';'
```

Il est par exemple équivalent d'écrire :

```
Station s1 = Station{};
s1.temperature = 2.3;
```

ou

```
s1 = Station{temperature=2.3;};
```

et l'on peut remarquer que si le type de la variable `s1` n'est pas précisé à gauche du `'='`, il est déduit de ce qui est indiqué à droite.

On fait appel aux services d'une entité à travers une expression de la forme :

```
[<variable>'=']<entité>'.'<service>'('[<argument>(,<argument>)*]')'';'
```

en reprenant les éléments des exemples précédents, on peut écrire un appel de service sur une instance `s1` de `Station` :

```
  tempf = s1.tempFahrenheight();
```

4.4 Relation et graphe d'interaction

Le mot clé `relation` est utilisé pour définir un type de graphe d'interaction. Il est ensuite possible de créer une ou plusieurs instances de graphes de ce type. Les concepts de relation et de graphe d'interaction dont il est question ici correspondent à ceux qui ont été définis de manière formelle en 3.3.3.

Une relation définit d'une manière générique les arcs d'un graphe.

Une propriété de relation est une valeur qui sera attachée à chaque arc du graphe. Les propriétés que l'on fait porter aux arcs sont des valeurs qui en principe n'ont un sens que lorsque plusieurs entités sont reliées les unes aux autres. La distance entre deux entités localisées dans l'espace par exemple n'a de sens que si l'on considère deux entités ensemble.

De la même façon un service de relation sera exécuté sur chaque arc du graphe, et un tel service n'est en principe utile que s'il porte sur plusieurs entités prises ensemble, comme par exemple un échange d'information entre une entité émettrice et une entité réceptrice.

4.4.1 Définition de relation

Une relation est définie par une expression de la forme suivante :

```
  'relation' <identifiant>'['<identifiant de rôle>(,<identifiant de rôle>)*']' '{'
    [définition de propriété]*
    [définition de service]*
 '}'
```

Les identifiants de rôles que l'on déclare dans la définition d'une relation ont une portée limitée au contenu de cette relation. Chaque rôle représente une entité. Accéder à une propriété d'un rôle, revient à accéder à la même propriété sur les entités qui joueront ce rôle dans le graphe d'interaction.

Exemple de définition d'une relation de voisinage :

```
relation Voisin[gauche,droit] {

  property real distance;

  service updateDistance() {
    dx = droit.posX - gauche.posX;
    dy = droit.posY - gauche.posY;
    distance = Math.sqrt(dx*dx + dy*dy);
 }
}
```

Dans cet exemple on fait usage de propriétés `posX` et `posY` aussi bien sur le rôle `gauche` que sur le rôle `droit`. Toute entité qui possède ces deux propriétés pourra être impliquée dans cette relation de voisinage et pourra jouer soit le rôle `gauche`, soit le rôle `droit`.

On voit ici que les deux rôles ne se distinguent que par leur nom, mais ils sont tout à fait symétriques. Il était cependant nécessaire de donner des identifiants distincts aux rôles dans la définition de la relation de façon à pouvoir éviter toute ambiguïté dans les opérations portants sur ces rôles. Avec la relation `voisin` définie ci-dessus, les rôles gauche et droit peuvent par exemple être affectés à un seul type d'entité. Une entité peut donc tout à fait jouer un rôle `gauche` sur un arc du graphe, et jouer le rôle `droit` sur un autre arc du graphe.

4.4.2 Instanciation et construction de graphe

Après avoir défini une relation, il est possible d'en créer une ou plusieurs instances. Il faut simplement spécifier un identifiant pour la variable qui va porter le graphe et indiquer les types d'entités qui vont jouer les différents rôles définis pour cette relation. Il est indispensable de fournir autant d'identifiants de types d'entités qu'il y a de rôles définis pour cette relation.

Dans le cas général, l'expression à utiliser est de la forme :

```
<identifiant de graphe>'='<relation>'['<type d'entité>(,<type d'entité>)+']'
                                     ['{'(<propriété>=<valeur>;)+'}']';'
```

Comme pour l'instanciation des entités, le type de la variable est déduit de ce qui est indiqué à droite du signe `'='`.

Exemple d'instanciation d'une relation :

```
gvoisins = Voisins[EntiteA,EntiteB];
```

La variable `gvoisins` que l'on déclare dans cet exemple va contenir un graphe d'interaction. On peut lui ajouter des instances d'entités et les connecter entre elles pour construire le graphe. On dispose des fonctions `connect()` et `disconnect()` pour ajouter ou retirer des arcs dans un graphe d'interaction.

Dans la suite de l'exemple précédent, on pourrait écrire :

```
a1 = EntityA{};
a2 = EntityA{};
b1 = EntityB{};
b2 = EntityB{};
b3 = EntityB{};
gvoisins.connect(a1,b2);
gvoisins.connect(a1,b3);
gvoisins.connect(a2,b1){distance = 10.3;};
```

Le dernier arc ajouté ici est en même temps initialisé avec une valeur pour sa propriété distance. Puisqu'un service permettant de mettre à jour cette propriété est disponible, une façon simple d'initialiser d'un seul coup cette propriété sur tous les arcs du graphe serait de faire un appel à ce service :

```
gvoisins.updateDistance();
```

Si l'on instancie la relation à nouveau par exemple en écrivant :

```
gvoisin2 = Voisins[EntiteC,EntiteA];
```

on aura un deuxième graphe distinct du premier mais que l'on pourra manipuler exactement de la même façon puisqu'il a les mêmes propriétés et les mêmes services.

4.4.3 Utilisation des graphes d'interactions

Une fois qu'un graphe est constitué, on peut consulter ou modifier sont état, en agissant sur les entités qu'il contient ou sur ses arcs.

L'action sur les entités, peut être effectuée soit de manière indépendante de la structure comme nous l'avons décrit en 3.3.4, soit en ne touchant que les entités qui sont les sommets des différents arcs du graphe, comme nous l'avons décrit en 3.3.5.

Interaction

On peut faire interagir les entités contenues dans un graphe en faisant simplement appel à l'un des services de la relation dont ce graphe est une instance.

La syntaxe d'appel d'un service sur un graphe d'interaction est de la forme :

```
<graphe>'.'<service>'('[<argument>(,<argument>)*]')'';'
```

Prenons par exemple une relation nommée `Surveillance` définie de la façon suivante :

```
relation Surveillance[capteur,recepteur] {
  service mesure(real seuil) {
    if (capteur.niveau > seuil) {
      recepteur.alerte(capteur.id);
    }
  }
}
```

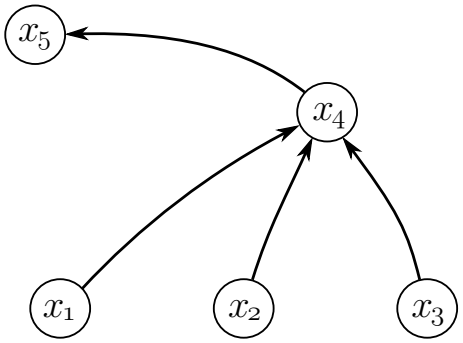


FIG. 4.1 – Illustration du graphe `reseau1`, instance de la relation `Surveillance`

Supposons que l'on ait construit une instance de la relation `Surveillance` qui prenne la forme d'un graphe nommé `reseau1`, comme celui de la figure 4.1.

Un seul appel au service `mesure()` :

```
reseau1.mesure(97.0);
```

aura pour effet de tester simultanément la propriété `niveau` des entités x_1,x_2,x_3 et x_4 pour les comparer au seuil indiqué et alerter le cas échéant les entités x_4 et x_5. Dans cet exemple les entités à l'origine d'un arc jouent donc le rôle `capteur`, et celles qui sont au bout de la flèche jouent le rôle `recepteur`. On remarquera que dans cet exemple, l'entité x_4 joue les deux rôles à la fois.

Accès aux entités d'un graphe indépendamment de sa structure

L'accès aux entités d'un graphe d'interaction doit permettre d'agir directement sur les propriétés de ces entités ou de faire appel à leurs services. Ces propriétés et services

sont connus à travers les rôles que jouent les entités du graphe. Ocelet permet d'accéder à toutes les entités jouant un rôle donné à travers une expression de la forme suivante :

```
<graphe>'['<rôle>']'
```

Si par exemple les entités jouant le rôle `capteur` possèdent une propriété booléenne `isActive`, il est possible d'affecter simultanément cette propriété sur tous les `capteurs` du graphe à travers l'instruction suivante :

```
reseau1[capteur].isActive=true;
```

Sélection d'arcs d'un graphe

L'opération de sélection sur un graphe à été définie formellement en 3.3.5. La syntaxe de cette opération prend la forme suivante :

```
<graphe>'?''('<expression logique>')'
```

Cette opération a pour effet de sélectionner tous les arcs du graphe pour lesquels l'expression logique est vérifiée. On peut utiliser les rôles de la relation dont le graphe est une instance dans l'expression logique. Il faut dans ce cas placer le nom du rôle entre crochets comme par exemple :

```
reseau1?([capteur].isActive == true)
```

Ce que l'on obtient à travers cette sélection est un sous-graphe du graphe de départ. Cet exemple donne un sous graphe de `reseau1` qui ne contient que les arcs dont la propriété `isActive` du rôle `capteur` vaut `true`. S'agissant d'un sous-graphe, on peut parfaitement lui appliquer les mêmes opérations que sur le graphe dont il est issu comme par exemple :

```
reseau1?([capteur].isActive == true).mesure(97.0);
```

Dans ce dernier exemple, le service `mesure()` ne sera appliqué que sur les arcs du sous-graphe sélectionné par l'expression logique.

Il est parfaitement possible de déclarer une variable de type graphe à partir d'une sélection avec une expression de la forme :

```
<identifiant de sous-graphe>'='<graphe>'?''('<expression logique>')'';'
```

ainsi on peut décomposer en deux instructions l'expression de l'exemple précédant en écrivant :

```
sousreseau1 = reseau1?([capteur].isActive == true);
sousreseau1.mesure(97.0);
```

Il est important de noter qu'un sous-graphe défini de cette façon ne constitue pas un graphe distinct de celui dont il est issu. Il s'agit seulement d'une sélection. Dans notre exemple `sousreseau1` et `reseau1` portent donc sur les mêmes entités, et si l'on change l'état du contenu de `sousreseau1`, cela change aussi l'état de `reseau1`.

4.4.4 Opérateur d'agrégation

Nous avons vu en 3.3.5 que lors de l'exécution d'un service de relation, il arrive que l'on se trouve dans une situation d'affectations multiples et simultanées :

- lorsqu'il y a partage d'une même propriété par des rôles différents,
- lorsque l'on a une multiplicité d'arcs incidents sur une entité.

Le concept d'opérateur d'agrégation, que nous avons proposé pour faire face à cette éventualité, a été intégré au langage Ocelet de façon à ce que les modélisateurs puissent définir ceux dont ils ont besoin, et les utiliser.

On définit un opérateur d'agrégation en décrivant comment choisir une valeur parmi un ensemble de valeurs candidates, ou en synthétisant une nouvelle valeur à partir du contenu de cet ensemble. Cela revient à définir une fonction qui prend en argument un groupe (non ordonné) de valeurs d'un type donné et qui renvoie une seule valeur de ce même type. Nous avons ajouté un mot clé en tête de cette définition de fonction pour identifier sans ambiguité qu'il s'agit d'un opérateur d'agrégation. Nous avons aussi ajouté la valeur actuelle de la variable qui doit être affectée, de façon à pouvoir en faire usage dans l'opérateur d'agrégation (il peut arriver que l'on veuille garder cette variable inchangée par exemple).

La syntaxe de définition d'un opérateur d'agrégation est de la forme :

```
'affect' <type de retour> <identifiant>'(' 'group['<type>']' <identifiant>','
                                          <type> <identifiant>')''{'
   <code de l'opérateur>
 '}'
```

le type de retour et le type des valeurs du groupe doivent donc être les mêmes.

Exemple de définition d'un opérateur d'agrégation qui renvoie simplement la moyenne des valeurs candidates :

```
affect real mean(group[real] gr, real val) {
    real sum = 0;
    for (v in gr) { sum = sum + v; }
    return sum / gr.size();
  }
```

La syntaxe d'utilisation d'un opérateur d'agrégation est une forme particulière d'affectation. Ces opérateurs ne sont en effet utilisés que dans des situations d'affectation à

l'intérieur d'un service de relation[14]. Cette syntaxe est de la forme :

```
<variable> '<='<opérateur d'agrégation>'=' <expression>';'
```

Exemple d'utilisation :

```
relation Rel[RoleA,RoleB] {

  service evolve(int coef) {
    RoleA.prop1 <=mean= RoleB.prop2 * coef;
    RoleB.prop2 = RoleB.prop2 +1;
  }
}
```

Dans le cas de cet exemple, la propriété `prop1` d'une entité jouant le rôle `RoleA` se verra affectée de la moyenne des valeurs calculées par l'expression `RoleB.prop2 * coef` provenant des différents arcs auxquels l'entité jouant le rôle `RoleA` est connectée.

On pourra remarquer que dans la définition de l'opérateur, un groupe est passé en argument, mais ce groupe n'apparait nulle part à l'utilisation. Ce groupe est en fait construit automatiquement par le moteur d'exécution d'Ocelet à partir des différentes valeurs candidates apparaissant lors de la simulation. La valeur actuelle de la variable affectée est elle aussi retrouvée par le moteur d'exécution qui s'occupe de la passer en argument.

Si aucun opérateur d'affectation n'est spécifié dans le service de la relation mais que l'on se retrouve dans une situation d'indétermination (si on avait écrit `RoleA.prop1 = RoleB.prop2 * coef ;` par exemple), c'est un opérateur de choix aléatoire prédéfini dans Ocelet qui est appliqué. Le choix d'une valeur prise au hasard dans le groupe est une garantie de pouvoir renvoyer une valeur du bon type dans tous les cas. Choisir par défaut un opérateur qui renvoie une moyenne (ou toute autre forme de calcul) n'aurait pas de sens si le groupe contient des valeurs de type texte ou booléen par exemple.

Les opérateurs d'agrégation ont été conçus pour lever l'indétermination dans certaines situations, mais on peut aussi s'en servir pour définir des règles de transition basées sur un voisinage. En effet les valeurs candidates réunies dans le groupe passé en argument proviennent des entités voisines selon la topologie du graphe d'interaction. Cela permet de décider comment une entité change d'état en fonction des états des entités voisines dans le graphe.

On peut prendre en exemple le Jeu de la vie [66], dont les règles pourraient être spécifiées dans un opérateur d'agrégation défini ainsi :

```
affect boolean LifeRule(group[boolean] voisins, boolean val) {
  int count=0;
```

[14] c'est d'ailleurs la raison pour laquelle nous avons choisi le mot clé `affect` dans la définition de ces opérateurs : il s'agit en quelque sorte de définir une forme spécifique d'affectation

```
  for (e in voisins) {
    if (e == true) {count = count+1;}
  }
  boolean result=val;
  if ((count < 2) || (count > 3)) {result = false;}
  if (count == 3) {result = true;}
  return result;
}
```

La relation portant la définition des interaction entre cellules voisines est alors particulièrement simple à écrire :

```
relation Life[centre,voisin] {
  service evolve() {
    centre.etat <=LifeRule= voisin.etat;
  }
}
```

Un seul appel au service `evolve()` de cette relation suffira à effectuer la collecte simultanée des états voisins de toutes les cellules du graphe, et appliquer la règle `LifeRule` qui affectera le nouvel état à chaque cellule.

4.5 Scenario

Un scenario est une suite ordonnée d'opérations. C'est à travers cette suite d'opérations que les divers constituants d'un modèle peuvent être construits, puis que l'on peut faire évoluer leurs états. Un scénario contient donc le programme proprement dit qui est exécuté lors d'une simulation. C'est par exemple dans un scenario que l'on instancie des entités et des relations pour construire des graphes d'interaction. C'est aussi dans un scenario que l'on place les appels aux services de graphes d'interaction, dans un ordre précis, pour faire évoluer ces graphes dans le temps.

La forme générale d'écriture d'un scénario est la suivante :

```
'scenario' <identifiant> {<code du scenario>}
```

Voici un exemple de scénario qui crée une instance de la relation `Surveillance` (définie en exemple dans la section précédente), construit le graphe de la figure 4.1, puis appelle une fois le service `mesure()` sur ce graphe :

```
scenario ReseauDeSurveillance {
  x1=Station{}; x2=Station{}; x3=Station{};
  x4=Station{}; x5=Station{};
```

```
  reseau1 = Surveillance[Station,Station];
  reseau1.connect(x1,x4);
  reseau1.connect(x2,x4);
  reseau1.connect(x3,x4);
  reseau1.connect(x4,x5);

  reseau1.mesure(97.0);
}
```

Ce scénario suppose bien entendu qu'un type d'entité nommé `Station` ait préalablement été défini, et qu'il possède bien les propriétés `niveau` et `id`, ainsi que le service `alerte()` lui permettant de jouer les rôles `capteur` et `recepteur` de cette relation.

Pour éventuellement simplifier l'écriture et la maintenance des modèles, il est possible de définir plusieurs scénarios dans un même modèle. C'est le scénario qui porte le même nom que le fichier principal du modèle qui sera considéré comme le point d'entrée pour les simulations. On pourra donc faire appel aux autres scénarios depuis ce scenario principal.

4.6 Datafacer

Ocelet est un langage métier qui ne comporte pas toutes les bibliothèques de manipulation de fichiers, d'accès à des bases de données, de fonctions mathématiques évoluées, de fonction graphiques, que l'on trouve en général associées aux langages généralistes. Ocelet étant traduit dans un langage généraliste cible, il est préférable de profiter directement des bibliothèques de fonctions disponibles pour ce langage cible. Ecrire ces bibliothèques entièrement en Ocelet serait non seulement fastidieux, mais surtout peu efficace parce que le langage n'est pas conçu pour cela.

Le mot clé *datafacer* a été construit par association des mots *data* et *interface*. Les datafacers servent principalement à procurer aux modèles Ocelet un accès à diverses formes de sources de données pour alimenter les modèles et exporter les résultats de simulations dans des formats exploitables par d'autres outils. Un modèle en Ocelet fera donc appel à un datafacer spécifique pour lire des fichiers dans un format donné, à un autre datafacer pour accéder au contenu d'une base de données puis éventuellement à un troisième pour exporter le résultat d'une simulation dans un format particulier.

Techniquement, les datafacers sont aussi une forme de passerelle entre le langage Ocelet et des bibliothèques du langage cible que l'on utilise. Il s'agit donc d'un outil aux usages multiples. Nous décrivons d'abord comment ces datafacers sont construits, puis nous montrons comment on peut les utiliser dans différentes situations.

4.6.1 Constitution d'un datafacer

Nous avons vu en 3.4.2 l'intérêt de considérer deux aspects dans l'accès à des sources de données : l'aspect technique, dépendant de la forme de stockage (format de fichier ou

schéma de base de données par exemple), et la construction d'éléments d'un modèle à partir de ces données. La structure d'un datafacer reprend ces deux aspects :

- une partie du datafacer est écrite directement dans le langage cible, pour être proche des différentes formes de sources de données et pour bénéficier des bibliothèques de fonctions de ce langage
- l'autre partie est écrite en Ocelet, pour offrir un accès en cohérence avec les modèles.

Lors de la construction d'un modèle avec Ocelet, on ne fait qu'utiliser des datafacers qui existent déjà. Les deux aspects d'un même datafacer sont sous forme de fichiers : un fichier portant l'extension `.oclt` qui contient la signature des services offerts par le datafacer, et un fichier portant l'extension `.jar` (pour la version actuelle d'Ocelet compilée en Java) qui contient les classes et méthodes contenant le code effectif de ces services.

Si l'on regarde par exemple un datafacer dédié aux fichiers de format *Shapefile*[15], la partie en Ocelet est un fichier nommé `Shapefile.oclt` dont un extrait est donné ci-dessous :

```
datafacer ShapeFile {
 service setFileName(text shp_name);
 service setCrsEPSG(text epsgCode);
 service view();
 service boolean hasNextRecord();
 . . .
}
```

et la partie en Java est un fichier nommé `datafacer_shapefile.jar`. Ce dernier contient une classe `Shapefile` avec des méthodes dont les signatures correspondent aux services déclarés dans `Shapefile.oclt`.

Le développement de nouveaux datafacers pour répondre à des besoins spécifiques est possible mais nécessite des compétences dans le langage cible.

4.6.2 Utilisation comme source de données

Pour utiliser un datafacer dans un modèle, on doit le déclarer à travers une expression de la forme :

```
'datafacer' <nom du datafacer>';'
```

Le nom dont il est question est celui du fichier portant l'extension `.oclt` mais sans cette extension.

[15] Il s'agit d'un format de fichier mis au point par la société ESRI qui est utilisé par de nombreux logiciels de traitement de l'information géographique et de cartographie.

Le modèle Ocelet voit ce datafacer comme étant une définition de type. Il faut donc créer au moins une instance pour pouvoir faire appel aux services du datafacer en s'appuyant sur cette instance.

Un modèle qui fait usage du datafacer `Shapefile` serait donc écrit de la façon suivante (extraits) :

```
datafacer Shapefile;
. . .

scenario Geomodele {
 . . .
  shp = ShapeFile{};
  shp.setFileName(shp_name);
  shp.setCrsEPSG("EPSG:2975");
 . . .
}
```

4.6.3 Utilisation comme bibliothèque de fonctions

Dans le cas où l'on utilise un datafacer non pas comme un lien vers des données mais comme une bibliothèque de fonctions, le principe est le même. Par exemple Ocelet fait appel à un datafacer nommé `Math` pour disposer des fonctions mathématiques usuelles. Voici un exemple de son usage :

```
datafacer Math;
. . .

  m = Math{};
  distance = m.sqrt(dx*dx + dy*dy);
. . .
```

On peut remarquer qu'un datafacer qui ne fait pas le lien avec des données ne porte aucun état. Forcer le modélisateur à créer une instance de ce type de datafacer avant de l'utiliser est une contrainte dont nous pourrions nous passer. Aussi nous envisageons à l'avenir de distinguer les datafacers qui font l'interface avec des données, des bibliothèques de fonctions qui correspondent plutôt à des objets statiques. Cela va impliquer l'ajout d'un mot clé distinct pour les bibliothèques de fonctions mais permettra de simplifier leur usage.

4.7 Autres éléments du langage Ocelet

4.7.1 Types de base

Les types de base d'Ocelet sont au nombre de quatre :

`boolean` Type de valeurs pouvant être soit vraie (`true`) soit fausse (`false`). Aux variables de type `boolean` on peut appliquer les opérateurs logiques d'algèbre de Boole : Négation, ET logique, OU logique, etc.

`int` Nombre entier relatif. Le domaine de valeurs est celui du type correspondant dans le langage cible dans lequel le modèle Ocelet est traduit, c'est à dire le type Integer de Java dans la version actuelle du compilateur. On peut appliquer aux nombres entiers les opérateurs classiques : addition, soustraction, multiplication, division.

`real` Nombres réels. Là encore les grandeurs que l'on peut exprimer sont limitées selon le type que l'on utilise dans le langage cible. Il s'agit du type Double de Java en l'occurence.

`text` Permet d'exprimer des variables avec un contenu textuel. On peut ensuite par exemple faire appel à l'instruction `print(variable)` pour afficher à l'écran le contenu d'une variable textuelle.

4.7.2 Structures

Le mot clé `structure` est utilisé dans Ocelet avec un sens proche des structures en langage C : cela permet de définir des propriétés composites sous la forme d'ensembles de valeurs de types différents. La définition d'une structure est une expression de la forme :

```
'structure' <identifiant> {
   <définition de propriété>+
 }
```

Exemple de structure permettant de représenter une position dans un espace à deux dimensions :

```
structure Position{
  property real x;
  property real y;
 }
```

Une fois définie, une structure peut être initialisée avec une syntaxe similaire à l'initialisation des entités. L'expression générale de l'initialisation est :

```
<identifiant de variable>'='<structure>'{'[<propriété>=<valeur>;]+'}'
```

et l'accès aux différentes propriétés internes d'une structure est une expression de la forme :

```
<structure>'.'<propriété interne>
```

Exemple d'usage de la structure `Position` définie plus haut :

```
pos1 = Position{ x=1250.45; y=3000.30;};
pos2 = Position{ x=1000.00; y=2920.00;};
dx = pos2.x - pos1.x;
dy = pos2.y - pos1.y;
```

Les structures contenant d'autres structures peuvent être construites. Comme par exemple :

```
structure Pos2D{
        property real x;
        property real y;
}

structure Pos3D{
        property Pos2D p2d;
        property real z;
}

  ...

  p2 = Pos2D{x=5.1; y=5.5;};
  p3 = Pos3D{p2d=p2; z=7.1;};
}
```

4.7.3 Collections prédéfinies : `group` et `list`

Ocelet offre la possibilité de construire des collections de valeurs qui sont toutes du même type sous deux formes différentes : `group` est qui un ensemble non ordonné, et `list` qui est une collection ordonnée. Les deux syntaxes de construction sont similaires :

```
<identifiant> '=' 'group['<type>']'';'
<identifiant> '=' 'list['<type>']'';'
```

On peut ensuite ajouter des valeurs à ces collections avec la fonction `add(<valeur>)`. Il faut noter que `group` est un ensemble, c'est à dire qu'il ne peut pas contenir deux fois le même élément. Voici un exemple qui illustre leur usage et leurs différences :

```
p = group[real];
p.add(5.1);
p.add(15.45);
p.add(2.2);
p.add(5.1);
```

```
q = list[real];
q.add(5.1);
q.add(15.45);
q.add(2.2);
q.add(5.1);

print("p : "+p);
print("q : "+q);
```

L'exécution de ce programme affiche le contenu des deux variables `p` et `q`, on obtient les résultats suivants :

```
p : [2.2, 5.1, 15.45]
q : [5.1, 15.45, 2.2, 5.1]
```

4.7.4 Instructions de contrôle

Les instructions de contrôle sont celles qui permettent d'agir sur l'ordre dans lequel s'enchainent les opérations d'un programme. Nous avons dans Ocelet :

- Les instructions de test qui orientent la suite de l'exécution selon qu'une expression logique est évaluée comme *vraie* ou *fausse*.
- Les instructions de boucle qui permettent de spécifier des suites d'instructions à exécuter de manière répétée. Deux formes d'instructions de boucle sont disponibles dans Ocelet : celles qui sont basées sur la gestion automatique d'un index, auxquelles nous ferons référence en tant que *boucles de type* ***`for`***, et celles dont la condition d'arrêt n'est pas forcément basée sur un index, auxquelles nous ferons référence en tant que *boucles de type* ***`while`***.

Conditions et opérateurs logiques

Une condition est une expression logique qui, comme les variables booléennes, peut être évaluée comme étant *vraie*, ou *fausse*. On fait en général appel à des conditions pour effectuer des comparaisons ou des tests. On dispose en Ocelet des mêmes opérateurs binaires que Java et C++ pour effectuer des comparaisons. Si `a` et `b` désignent deux expressions :

`a == b` → *vrai* si `a` est égal à `b`
`a != b` → *vrai* si `a` est différent de `b`
`a < b` → *vrai* si `a` est inférieur `b`
`a <= b` → *vrai* si `a` est inférieur ou égal à `b`
`a > b` → *vrai* si `a` est supérieur `b`
`a >= b` → *vrai* si `a` est supérieur ou égal à `b`

Le caractère `'!'` fait usage d'opérateur unaire de négation, c'est à dire que si `a` est *vrai*, alors `!a` est *faux*.

Des opérateurs permettent de combiner entre elles plusieurs conditions. Ces opérateurs sont le *OU* logique noté `||` et le *ET* logique noté `&&`. Si `v` est une condition *vraie* et `f` une condition *fausse*, leurs combinaisons à l'aide de ces opérateurs produiront les résultats suivants :

`v || v` → *vrai*	`v && v` → *vrai*
`v || f` → *vrai*	`v && f` → *faux*
`f || v` → *vrai*	`f && v` → *faux*
`f || f` → *faux*	`f && f` → *faux*

Instructions de test

Une instruction de test permet de n'exécuter une portion de programme que si une condition est *vraie*, elle permet aussi d'indiquer quelle portion de programme doit être exécutée dans le cas ou cette condition est *fausse*.

La syntaxe est de la forme :

```
'if' '('<condition>')' '{' <bloc d'instructions> '}'
 ['else' '('<condition>')' '{' <bloc d'instructions> '}']
```

Exemple d'usage :

```
if ((temperature > seuilMin) && (temperature < seuilMax)) {
 seeds.evolve();
}
else { AlerteTemperature(temperature); }
```

Boucles de type `for`

Les boucles de ce type assurent l'incrémentation automatique d'un index. Elles sont à privilégier lorsque l'on connait à l'avance le nombre de répétitions à effectuer.

Leur syntaxe peut prendre deux formes différentes :

1. Première forme, dans laquelle l'index prend successivement les valeurs contenues dans un ensemble

   ```
   'for' '('<variable d'index> 'in' <ensemble de valeurs>')'
     '{' <bloc d'instructions> '}'
   ```

   La variable d'index est initialisée avec la première valeur de l'ensemble fourni. Le bloc d'instructions est ensuite exécuté, dans lequel il est possible de faire usage de

cette variable d'index. La variable d'index tente ensuite de prendre une valeur suivante de l'ensemble. Si toutes les valeurs de l'ensemble ont déjà été parcourues, la boucle s'arrête, dans le cas contraire un nouveau tour de boucle est entamé avec une nouvelle valeur dans la variable d'index.

L'ensemble de valeurs peut être spécifié de deux manières différentes :

(a) Sous la forme d'un intervalle `<valeur minimale> .. <valeur maximale>`. Exemple qui affiche à l'écran une série d'entiers de 1 à 30 :

```
for (i in 1 .. 30) {print(i);}
```

(b) Par un identifiant de structure de données représentant un ensemble. Les structures de données `list` (4.7.3) et `group` (4.7.3) correspondent à des ensembles de valeurs, respectivement ordonnés et non ordonnés.

Exemple d'usage d'une instruction de boucle `for` indexée par une liste pour afficher les jours de la semaine :

```
property list[text] semaine;
 ...

semaine.add(" lundi ");
semaine.add(" mardi ");
 ...
semaine.add(" dimanche ");

 ...

for (jour in semaine) {print(jour);}
```

Le fait d'avoir utilisé la structure `list` dans cet exemple permet d'être sûr que les jours seront parcourus dans l'ordre d'initialisation de l'ensemble par l'instruction `for`. Si l'on utilisait `group` à la place de `list`, cet ordre ne serait plus garanti.

On remarquera l'inférence de type qui est effectuée par Ocelet pour la variable d'index de la boucle. Il n'est en effet pas nécessaire de déclarer le type de la variable d'index, celui-ci est automatiquement le même que le type des valeurs contenues dans l'ensemble fourni, qu'il s'agisse d'une structure de données ou d'un intervalle.

2. La seconde forme permet de spécifier précisément une expression logique qui sera évaluée comme critère d'arrêt, et la façon dont l'incrémentation de la variable d'index doit être effectuée.

```
'for' '('<initialisation>';' <condition>';' <incrémentation>')'
  '{' <bloc d'instructions> '}'
```

L'initialisation permet de définir une variable d'index. A chaque tour de boucle le bloc d'instructions est exécuté, puis l'incrémentation est effectuée, enfin la condition est évaluée. Si cette expression est *fausse*, la boucle s'arrête là, si au contraire elle est *vraie*, alors un nouveau tour de boucle commence.

Pour permettre la comparaison avec la première forme, reprenons l'exemple qui affiche dans la console une série d'entiers de 1 à 30 :

```
  for (int i=1; i<=30; i++) {print(i);}
```

Boucles de type `while`

Ces boucles répètent un bloc d'instructions tant qu'une condition d'arrêt n'est pas *vraie*. Il y a là aussi deux formes disponibles selon que l'on souhaite tester la condition d'arrêt avant l'exécution du bloc d'instructions ou après.

1. Première forme, dans laquelle la condition d'arrêt est évaluée d'abord. La syntaxe prend la forme suivante :

   ```
   'while' '('<condition>')' '{' <bloc d'instructions> '}'
   ```

   Exemple dans lequel on est obligé d'initialiser à l'avance la variable `temperature` qui doit être évaluée dans la condition d'arrêt, avant l'exécution du bloc d'instructions. Cela signifie que si la condition s'avère *vraie* dès le départ, le bloc d'instructions ne sera jamais exécuté.

   ```
   temperature = thermometre.getTemp();
   while (temperature > 0) {
     seeds.evolve();
     temperature = thermometre.getTemp();
   }
   ```

2. Seconde forme, où la condition d'arrêt est évaluée après l'exécution du bloc d'instructions.

   ```
   'do' '{' <bloc d'instructions> '}' 'while' '('<condition>')'
   ```

   L'exemple suivant est proche du précédent mais on voit ici que la variable `temperature` utilisée dans la condition d'arrêt n'a pas besoin d'être initialisée à l'avance. Cela signifie cependant que le bloc d'instructions sera exécuté au moins une fois quelle que soit la valeur de la condition d'arrêt.

   ```
   do {
     seeds.evolve();
     temperature = thermometre.getTemp();
   } while (temperature > 0)
   ```

4.7.5 Fonctions d'écriture

Deux fonctions d'écriture ont été prédéfinies dans le langage pour permettre les opérations de base que sont l'affichage de texte à l'écran, et l'écriture de texte dans un fichier.

On peut noter qu'il n'y a pas de fonction de lecture correspondante, l'utilisation de datafacer étant privilégiée pour cela.

Affichage de texte sur la console

L'instruction `'print' '('<expression>')'` permet l'affichage d'une expression à l'écran. L'expression peut être un texte directement spécifié entre guillemets, ou tout autre expression pouvant être transformée en texte. En particulier si l'on indique un nom de variable, Ocelet va tenter de transformer cette variable en texte pour pouvoir l'afficher. Le caractère `'+'` peut servir à concaténer des expressions textuelles.

Exemple d'expressions correctes :

```
print("Ocelet");
print("Mon nom est "+nom);
print("Valeur de la case ("+i+","+j+"):"+monEntite.getCase(i,j))
   ;
```

Ecriture de texte dans un fichier

Pour écrire une ligne de texte dans un fichier, la syntaxe est la suivante :

```
'print2file' '('<nom de fichier> ',' <ligne de texte>')'
```

Chaque appel à `print2file` ajoute une ligne à la fin du fichier indiqué.

On peut remarquer qu'il n'y a aucune instruction pour ouvrir ou fermer un fichier comme c'est le cas dans de nombreux langages. Le fichier est en fait ouvert puis refermé automatiquement à chaque appel. Nous n'avons pas recherché l'efficacité mais plutôt la simplicité d'utilisation. Il est bien sûr toujours possible de faire appel à un datafacer spécialisé si l'on désire privilégier l'efficacité ou si l'on a besoin de produire des formats de fichier différents.

Exemple d'utilisation :

```
text f = "Results.csv";
for (i in 1 .. 2000) {
 print2file(f,i + "," + monEntite.getVal(i));
}
```

Le code de cet exemple enregistre 2000 lignes dans le fichier `Results.csv`, chaque ligne étant constituée de deux valeurs séparées par une virgule. Ces valeurs ont été transformées au format texte et le fichier résultant ne contient que du texte.

4.7.6 Inclusion de modèles : `uses`

Pour favoriser une construction modulaire de modèles, et faciliter la réutilisation de code, nous avons ajouté une instruction `uses` permettant d'importer dans un modèle des fichiers source Ocelet disponibles par ailleurs. La syntaxe d'utilisation de cette instruction est la suivante :

```
['uses' <fichier ocelet>]*
```

Le fichier source Ocelet indiqué est alors importé au moment de la compilation et il sera compilé en même temps de le modèle principal. Un fichier source importé de cette façon est donc traité comme si son contenu faisait partie du fichier source principal.

4.8 Conclusion

Nous avons construit un outil de modélisation et de simulation sous la forme d'un langage métier nommé *Ocelet*. Le fait qu'il s'agisse d'un langage traduit notre volonté d'offrir une grande capacité d'expression aux modélisateurs, et l'aspect métier se traduit par trois aspects :

1. Une syntaxe allégée par rapport à des langages de programmation généralistes. Cet outil s'adresse en priorité à des chercheurs ou techniciens qui ont des notions de programmation mais qui ne sont pas informaticiens. C'est la raison pour laquelle nous avons tenté de limiter autant que possible le nombre de mots clés, et nous avons essayé de faciliter la vie du modélisateur à chaque fois que cela est possible (avec de l'inférence de type par exemple).
2. Les principaux concepts que nous avons imaginé pour construire et manipuler des graphes d'interaction sont représentés sous la forme de mots clés du langage, et la sémantique associée à ces concepts est intégrée directement dans le moteur d'exécution d'Ocelet.
3. Des moyens sont offerts (à travers des *datafacers* fournis) pour manipuler toutes sortes de sources de données avec peu d'effort. Il s'agit en particulier des différents formats de stockage et manipulation d'information géographique (Shapefile, SGDB-Spatial, conversion automatique de système de coordonnées, KML, etc.) et d'aide à la construction de graphes de voisinage de toutes sortes.

Les éléments de langage présentés ici sont ceux que nous avons suffisamment bien définis pour être implémentés. La majorité d'entre eux a d'ailleurs déjà fait l'objet d'une implémentation qui a permis de construire les modèles exposés dans le chapitre 6. Ce travail de définition du langage n'est cependant pas terminé et va se poursuivre au delà de nos travaux de thèse. Par exemple nous n'avons pas encore défini de syntaxe pour les fonctions d'analyse globale et locale de graphes (ensemble de fonctions décrites en 3.3.6).

L'usage du langage a aussi fait apparaître des améliorations possible. C'est par exemple le cas pour les datafacers qui remplissent actuellement des fonctions de différentes natures.

Ils servent d'une part à faire le lien entre diverses formes de sources de données et les modèles, et d'autre part à bénéficier de la richesse des bibliothèques disponibles du langage cible. Il serait peut-être préférable de spécialiser ces deux aspects de leur fonction à l'avenir. Cela nécessite l'introduction de nouveaux mots-clés pour distinguer chaque usage : *datafacer* pour faire le lien avec des données, et par exemple *library* pour les bibliothèques de fonctions n'ayant pas de lien précis avec des données (comme la bibliothèque `Math` qui donne accès aux fonctions mathématiques usuelles).

Associés à ce langage, un compilateur, un moteur d'exécution et un environnement de développement ont été produits. Il s'agit de travaux d'ingénierie complémentaires aux travaux de thèse, sans lesquels nous n'aurions pas pu expérimenter réellement la construction de modèles et leur simulation à l'aide d'Ocelet. Nous donnons dans le chapitre suivant un aperçu de ces outils qui ont donné vie à notre langage.

Chapitre 5

Outillage d'Ocelet

Sommaire

5.1 Introduction

La construction du langage Ocelet a été menée avec l'objectif de rendre cet outil réellement opérationnel. Un important travail d'ingénierie a été fourni pour outiller le langage et ce travail va se poursuivre au delà de nos travaux de thèse. C'est donc un état en évolution de ces outils qui est présenté ici, c'est à dire une première version expérimentale qui nous permet de tester les concepts sur lesquels Ocelet est basé. L'implémentation des différents modules n'a pas encore fait l'objet d'optimisations, nous avons surtout cherché à construire un prototype qui respecte les principes, la sémantique et la syntaxe décrits dans les chapitres 3 et 4.

Ocelet est un langage métier compilé, c'est à dire que les programmes écrits en Ocelet sont traduits dans un langage généraliste avant de pouvoir être exécutés. Dans sa version actuelle, c'est en langage Java que les programmes écrits en Ocelet sont traduits. Le code généré en Java fait appel à un ensemble de classes développées par ailleurs, qui constituent le *moteur d'exécution*[16]. Cela signifie que la sémantique du langage est implémentée en

[16]traduction personnelle du terme d'usage anglais *runtime*

partie dans le générateur de code, et en partie dans le moteur d'exécution.

Un modèle écrit en Ocelet peut faire appel à un ou plusieurs datafacers. Chaque datafacer est constitué de deux parties : une partie en Ocelet qui contient la signature des fonctions qu'il fournit, et une partie sous la forme de classes java qui contiennent le code effectif de ces fonctions. La partie en Ocelet des datafacers doit être fournie au compilateur pour que celui-ci ait connaissance des signatures des fonctions utilisées dans le modèle.

Les classes générées par le compilateur sont dépendantes des classes du moteur d'exécution, et des classes des différents datafacers utilisés dans le modèle. L'ensemble de ces classes constitue le modèle exécutable qui permet d'effectuer des simulations (figure 5.1).

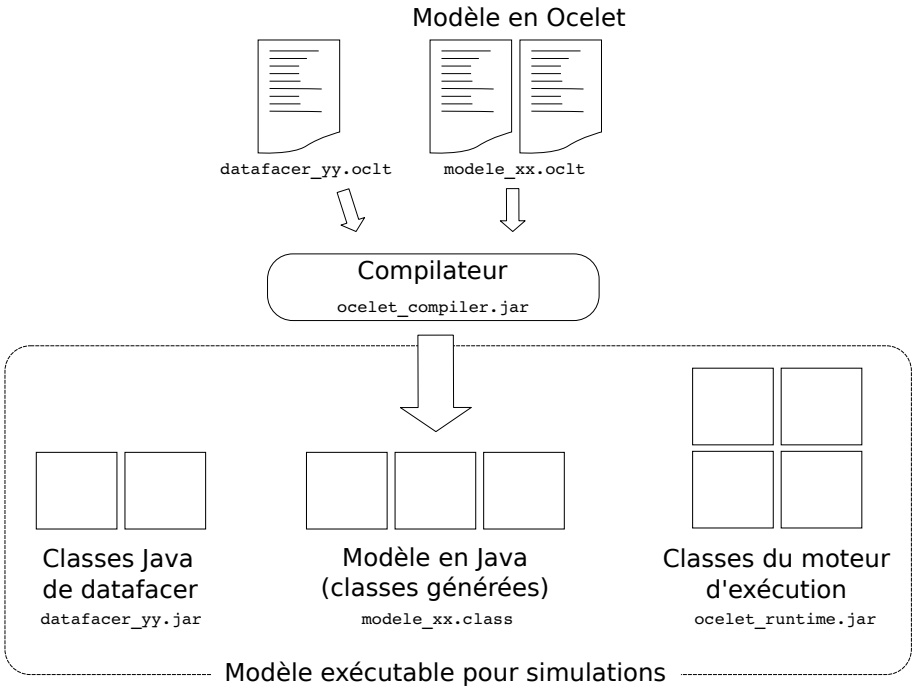


FIG. 5.1 – Eléments de construction d'un modèle de simulation

Nous avons également construit un environnement de développement pour Ocelet qui est basé sur un composant de la plate-forme Eclipse [1]. Cet environnement permet la création et la gestion de projets de modélisation, la compilation de modèles, et l'exécution de simulations. En cas de besoin, cet environnement peut produire une version compacte d'un modèle exécutable sous la forme d'un fichier archive de Java (jar) qui contient les classes générées, celles des datafacers et de leurs éventuelles dépendances, et celles du moteur d'exécution.

Sont présentés dans ce chapitre les trois principaux outils construits pour Ocelet : le compilateur, le moteur d'exécution, et l'environnement de développement.

5.2 Compilation et génération de code

Les compilateurs de langages de programmation généralistes sont en général écrits à l'aide du langage qu'ils doivent compiler. C'est une forme d'auto-amorçage[17]. Comme la plupart des langages métiers, Ocelet n'est pas conçu pour la compilation, et le compilateur d'Ocelet a donc été écrit à l'aide d'un langage généraliste, Java en l'occurrence.

L'architecture du compilateur d'Ocelet est assez classique :
- Un analyseur lexical (*lexer*) lit un fichier source de modèle Ocelet et en extrait une série d'unités lexicales.
- Un analyseur syntaxique (*parser*) construit la structure syntaxique du programme à partir des unités lexicales. Cette structure prend une forme arborescente (figure 5.3), c'est l'arbre de syntaxe abstraite (AST[18]). Les nœuds de cet arbre sont des instances de classes qui représentent les différents types d'éléments que l'on peut trouver dans le langage. L'AST est une représentation logique du programme que l'on souhaite compiler.
- Un analyseur sémantique parcours l'AST, il effectue un contrôle de typage, enrichit l'AST pour préparer la génération de code, et génère une table des symboles.
- Un générateur de code parcourt l'AST et génère du code correspondant au contenu de chaque nœud de l'arbre dans un langage cible.

Nous avons utilisé le compilateur de compilateur Tatoo développé à l'université de Marne-La-Vallée [32, 4] pour produire de manière automatique certains éléments du compilateur d'Ocelet. La grammaire d'Ocelet[19] est décrite sous la forme d'une liste de règles syntaxiques exprimées dans le format *ebnf*[20]. A partir de cette grammaire, Tatoo génère l'analyseur lexical, l'analyseur syntaxique, et un ensemble de classes Java spécifiques qui correspondent aux types des nœuds de l'AST (figure 5.2).

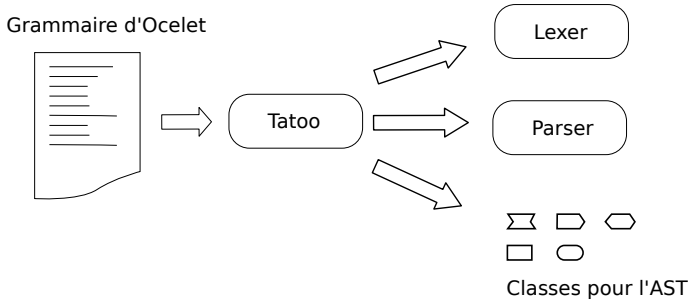


FIG. 5.2 – Le compilateur de compilateur Tatoo génère une partie du compilateur d'Ocelet

Une particularité intéressante des nœuds de l'AST générés par Tatoo est qu'ils ac-

[17] le terme couramment utilisé est *bootstrap*
[18] AST : Abstract Syntax Tree
[19] La grammaire d'Ocelet est consultable en annexe de ce document(7.2)
[20] EBNF : Extended Backus-Naur Form

ceptent les visiteurs (nous faisons ici référence au patron de conception *Visiteur* [64]). Ainsi, l'analyse sémantique, et le générateur de code Java sont réalisés sous la forme de visiteurs : pour chaque type de nœud nous avons écrit une méthode `visit(...)` qui effectue le traitement correspondant à ce nœud lors du parcours de l'AST (figure 5.4). La classe `GenVisitor`[21] par exemple contient les méthodes `visit(...)` du générateur de code. Chacune de ces méthodes traduit un type d'élément de modèle Ocelet (définition de Relation, déclaration de variable, appel de service, etc.) en code Java équivalent. Dans certains cas le code Java généré effectue telle quelle l'opération, dans d'autres cas, il fait appel à des classes du moteur d'exécution. Notons enfin que le générateur de code peut

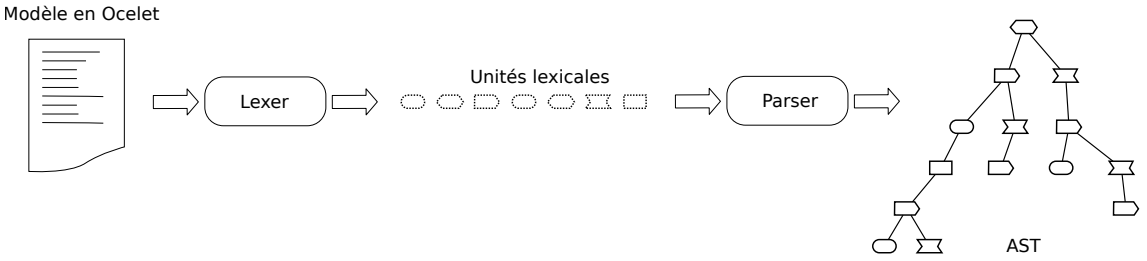


FIG. 5.3 – Compilation : phase d'analyse et construction de l'AST

produire directement des fichiers Java compilés (avec l'extension `.class`), mais il peut aussi produire des fichiers sources de Java (avec l'extension `.java`) pour le cas où l'on souhaite consulter le code qui a été produit.

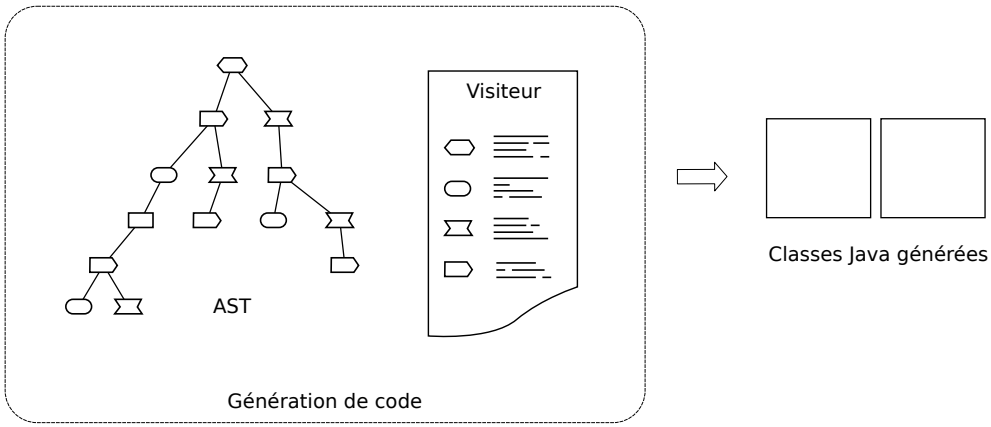


FIG. 5.4 – Compilation : phase de génération de code à l'aide de visiteurs.

Pour faciliter l'usage et la diffusion du compilateur, l'ensemble des outils qui le constituent (Tatoo, et les différents modules que nous avons développés) sont empaquetés dans un seul fichier archive : `ocelet-compiler.jar` dont la version la plus récente est disponible sur le serveur Subversion d'Ocelet à l'adresse suivante :
`http ://svn.ocelet.fr/Compiler/lib/ocelet-compiler.jar`.

[21] `GenVisitor.java` peut être consulté sur `http ://svn.ocelet.fr/Compiler/src/fr/ocelet/compiler/gen/`

Un modélisateur qui souhaite utiliser ce compilateur pourra le faire avec une commande du type :
`java -jar ocelet-compiler.jar monModele.oclt`

Une autre façon d'utiliser ce compilateur, sans passer par une ligne de commande, consiste à faire appel à l'environnement de développement décrit en section 5.4 de ce chapitre.

5.3 Moteur d'exécution

Chaque modèle écrit en Ocelet est différent et contient des définitions de Relations, d'Entités, de Scenario dont le contenu reflète les intentions du modélisateur. Mais ces modèles ont en commun un certain nombre de qualités et de comportements qui correspondent à la sémantique fondamentale d'Ocelet. Il s'agit en particulier de la façon dont les graphes d'interaction changent d'état, de la gestion de l'historique des propriétés dans les entités, du fonctionnement des opérateurs d'agrégation, ou encore des mécanismes de sélection sur les graphes. Tous ces comportements étant communs aux modèles, il n'est pas nécessaire de les produire par génération de code, nous les avons implémentés sous la forme d'un ensemble de classes Java qui constituent le moteur d'exécution d'Ocelet. Ces classes sont regroupées sous la forme d'un seul fichier archive Java : `ocelet_runtime.jar` pour faciliter leur déploiement.

Pour que les classes générées par le compilateur bénéficient de ces comportements de base, nous avons fait en sorte que la plupart d'entre elles héritent de classes abstraites du moteur d'exécution. Dans sa version actuelle, le moteur d'exécution contient surtout des classes relatives au fonctionnement des entités et des relations, il n'y a rien à propos des scénarios qui sont dans leur intégralité constitués de code généré à la compilation. Nous détaillons dans la suite de cette section quelques classes du moteur d'exécution pour donner un aperçu de son fonctionnement et la façon dont le code généré en fait usage.

5.3.1 Entités et propriétés

Prenons l'exemple d'une définition d'entité en Ocelet :

```
entity Pond {
  property real level;
  property text name;
  service printLevel() {
    print("Niveau de "+name+": "+level);
  }
}
```

Le code Java généré par le compilateur sera le suivant :

```
public class Pond extends AbstractEntity {

  public Pond() {
    super();
    defProperty("level",new Hproperty<Double>());
    defProperty("name",new Hproperty<String>());
  }

  public void printLevel() {
    System.out.println("Niveau pour " +
          (String)getProperty("name") + ": " + (Double)getProperty
             ("level"));
  }
}
```

La classe `AbstractEntity` est une classe abstraite faisant partie du moteur d'exécution. Le compilateur a produit une classe `Pond` qui étend `AbstractEntity` et qui hérite donc de son comportement.

Au lieu de générer simplement quelque chose comme `Double level ;` pour la propriété `level`, on utilise la méthode `defProperty()` de `AbstractEntity` pour ajouter cette propriété à l'entité `Pond`. Nous avons en effet besoin de gérer les propriétés sous la forme de listes pour pouvoir appliquer un mécanisme transactionnel sur toutes les propriétés lors des interactions dans un graphe. De plus le fait de traiter tous les types de propriétés sous la forme d'une classe paramétrée (`Hproperty`) nous permet de doter ces propriétés d'un certain nombre de fonctions utiles.

Nous avons vu en 3.3.5 qu'il existe des situations où une propriété pourrait être affectée simultanément par plusieurs valeurs différentes. Pour faire un choix dans ce type de situation nous avons proposé la possibilité de définir un opérateur d'agrégation qui peut être assigné à la propriété. Une propriété donnée ne peut avoir qu'un seul opérateur d'agrégation à la fois, mais cet opérateur peut être changé selon les besoins. La classe `Hproperty` possède une référence vers un objet de type `AffectOperator`.

`Hproperty` est une classe paramétrée par un type qui est spécifié lors de l'instanciation. `AffectOperator` est une interface qui elle aussi est paramétrée, par le même type que la propriété à laquelle elle est reliée. On peut voir dans la définition de `Hproperty.java` (ci-dessous) les différents éléments qui servent à implémenter ces opérations d'agrégation :

```
public class Hproperty<T> implements Iterable<T> {
  ArrayList<T> content;
  Group<T> future;
  boolean containsFuture;
  int historySize;
  AffectOperator<T,Group<T>> ao;
```

```
  ...
```

L'état d'une propriété est contenu dans la variable `content`. Cette variable est en fait une liste qui peut contenir la valeur actuelle à un moment donné mais aussi l'historique des valeurs passées avec une profondeur qui est précisée par `historySize`.

La variable `future` est chargée de contenir toutes les valeurs candidates pour le nouvel état de cette propriété. La variable `containsFuture` permet de savoir s'il est nécessaire de faire ou non appel à l'opérateur d'agrégation pour décider du nouvel état. L'opérateur d'agrégation est doté d'une méthode `compute()` permettant de faire ce choix, elle est définie comme suit dans `AffectOperator` :

```
public interface AffectOperator<T,R extends Collection<T>> {
  public T compute(R future, T actualValue);
}
```

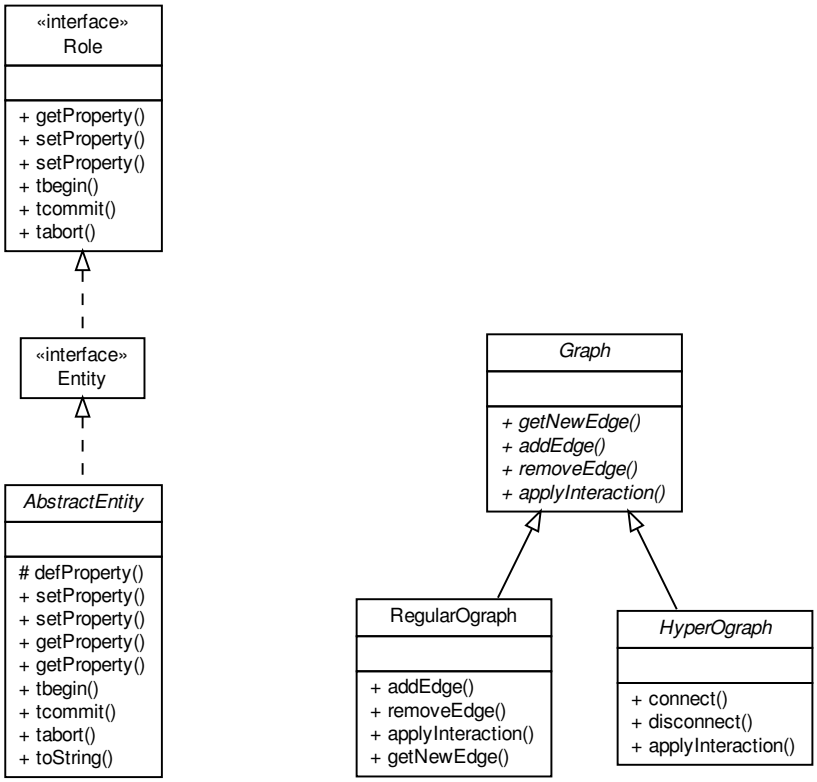
La valeur renvoyée par cette méthode est calculée à partir de l'ensemble des valeurs candidates (prises dans `future` qui est un groupe de valeurs), et de l'état actuel (`actualValue`).

Par défaut, c'est l'opérateur d'agrégation `Any` qui est appliqué. Cet opérateur choisit une valeur prise au hasard parmi les valeurs candidates. C'est en effet un moyen sûr pour renvoyer une valeur du bon type quelque soit le type de la propriété.

La collecte des différentes valeurs candidates pour un changement d'état, et le moment où l'on fait appel à l'opérateur d'agrégation, sont gérés par un mécanisme transactionnel implémenté dans les classes du moteur d'exécution relatives aux graphes d'interaction que nous exposons dans la section suivante.

5.3.2 Graphes

Le concept de graphe d'interaction a été décrit dans le chapitre 3 de la manière la plus générale possible, c'est à dire en se basant sur un nombre de rôles qui n'est pas limité à deux. Si l'on s'en tient à cette définition, tous les graphes d'interaction contruits avec Ocelet sont potentiellement des hypergraphes. On peut cependant s'attendre à ce qu'une bonne proportion de modèles fassent usage de graphes ne comportant que deux rôles. Nous avons séparé les graphes des hypergraphes dans le moteur d'exécution pour pouvoir leur appliquer des traitements différents. La classe `RegularOgraph` représente les graphes portant sur deux rôles, la classe `HyperOgraph` représente les hypergraphes, et la classe abstraite `Graph` est une généralisation des deux premières. D'autres formes de graphes pourront venir enrichir le moteur d'exécution par spécialisation de la classe `Graph` ou de ses sous-classes. Nous pensons en particulier à des graphes dont la topologie est connue à l'avance et sur lesquels il est possible de proposer des algorithmes de traitements plus



(a) (b)

FIG. 5.5 – Diagrammes de classes UML relatifs aux entités (a) et graphes d'interaction (b) du moteur d'exécution

efficaces, optimisés pour cette topologie.

Les méthodes communes à tous les types de graphes et qui donc se trouvent dans `Graph` sont relatives à la construction et la suppression d'arcs, mais aussi à l'application d'une interaction sur l'ensemble du graphe. Nous allons détailler plus particulièrement ce mécanisme d'interaction et le système de transaction qui lui est associé.

La classe `Graph` définit la méthode suivante :

```
  public abstract void applyInteraction(Interaction gi);
```

Sa fonction consiste à prendre une définition d'interaction, et à appliquer cette interaction sur tous les arcs du graphe simultanément. Cette méthode est définie de manière abstraite et elle peut être implémentée différemment dans `RegularOgraph` et dans `HyperOgraph`.

`Interaction` est une interface, qui définit simplement une méthode permettant de faire interagir un nombre fini de rôles les uns avec les autres. Cette interface correspond au concept de fonction d'interaction (3.3.2), elle correspond aussi au `service` d'une relation d'Ocelet (4.4) :

```
public interface Interaction {
  public <T extends Collection<Role>> void interact(T roles);
}
```

Les rôles que l'on passe à cette fonction `interact(T roles)` peuvent correspondre à n'importe quelle entité. Cela est possible parce que nous avons fait en sorte de définir une interface `Role` que la classe `AbstractEntity` implémente (figure 5.5(a)).

Pour chaque service défini dans une relation, le compilateur génère une classe qui implémente l'interface `Interaction`. Chacune de ces classes contient donc la fonction `interact(T roles)` dont le contenu est une traduction en Java de ce que le modélisateur a spécifié en Ocelet.

Lors de l'appel à un service sur le graphe d'une relation, on effectue deux parcours du graphe :

Premier parcours du graphe : sur chaque arc visité on effectue deux opérations

1. On Signale un début de transaction aux entités de chaque extrémité de l'arc. Cela a pour effet d'indiquer à ces entités que leur état ne devra pas être modifié avant la fin de la transaction. Chaque opération affectant une propriété aura pour effet d'ajouter une nouvelle valeur à la liste de la variable `future` de cette propriété. Chaque opération de lecture d'une propriété renverra la valeur présente avant le début de la transaction. Autrement dit, lors d'une transaction, on lit l'état n des propriétés, et on écrit l'état $n + 1$.
2. On appelle la fonction `interact(T roles)`. Cette fonction peut contenir des opérations de lecture ou de modification de valeurs de propriétés des entités aux extrémités de l'arc visité. Si une entité se trouve à l'extrémité de plusieurs

arcs, elle pourra être affectée d'un nouvel état plusieurs fois durant le parcours du graphe, et chaque nouvelle valeur sera simplement ajoutée à la liste de la variable `future` des propriétés concernées.

Second parcours du graphe : sur chaque arc visité, on signale aux entités la fin de la transaction. Cela a pour effet de changer effectivement l'état des entités. Lorsque la variable `future` d'une propriété ne contient qu'une seule valeur, celle-ci est simplement prise comme nouvel état. Si `future` contient plusieurs valeurs, le nouvel état sera le résultat d'un appel à l'opérateur d'agrégation de la propriété.

Pour signaler les débuts et fin de transaction aux entités, ce sont les fonctions `tbegin()` et `tcommit()` (figure 5.5(a)) auxquelles on fait appel. Les opérations de lecture et d'écriture sur les propriétés d'entités se font à travers des fonctions `setProperty()` et `getProperty()` qui respectent le comportement transactionnel que nous venons de décrire.

5.4 Environnement de développement

Pour faciliter la programmation de modèles de simulation avec Ocelet, nous proposons un environnement de développement nommé Ocelet Modelling Platform (OMP). Cet environnement regroupe en une seule interface utilisateur homogène les différentes fonctions dont un modélisateur a besoin pour travailler avec Ocelet : création et maintenance de modèles, édition de source Ocelet, compilation, lancement de simulation, affichage ou exportation de résultats de simulations, et en cas de besoin l'accès au code Java généré par le compilateur.

L'OMP est construit à partir du module RCP (Rich Client Platform) d'Eclipse [2]. Eclipse RCP propose un ensemble de modules de base permettant la construction d'un environnement graphique homogène pour toutes sortes d'applications :

- De riches bibliothèques d'éléments d'interface graphique.
- Un environnement intégrant différentes vues, avec une gestion souple de la mise en page à travers des onglets, un partage de l'espace de travail, et le regroupement d'ensembles de vues par thèmes (les *perspectives*).
- Une gestion intégrée des différents réglages de personnalisation de l'environnement à travers des boites de dialogue de préférences.
- Un mécanisme d'extension par ajout de modules (*plugins*) qui respecte les protocoles de l'OSGi[22] ce qui facilite les évolutions et mises à jour.

5.5 Version distribuée

Ocelet a aussi servi à expérimenter une approche distribuée de modélisation. Dans la suite de travaux sur les modèles à base de composants auxquels nous avions participé [39], A. Ait Lahcen et al. [9] ont proposé un moteur d'exécution et un compilateur d'Ocelet produisant un composant indépendant pour chaque élément d'un modèle. Ces composants

[22] OSGi : Open Services Gateway initiative [3].

sont dotés d'interfaces qui leur confère des fonctions que l'on trouve classiquement dans les architectures orientées services (publication, découverte et couplage de services), mais qui permet aussi la composition dynamique de services, et le contrôle d'exécution des services pour gérer les flux de données. Un intérêt de cette approche est de séparer la partie utile (au sens de la modélisation), de la mécanique de gestion des composants. Le modélisateur peut ainsi se concentrer sur son modèle et ne pas se soucier des détails d'implémentation liés à la nature distribuée du système qui est généré.

Nous ne développons pas dans le détail ces travaux dans ce document, ils font en effet l'objet d'une partie de la thèse d'Ayoub Ait Lahcen que le lecteur intéressé est invité à consulter.

5.6 Conclusion

Nous avons l'ambition de produire un outil de modélisation réellement opérationnel, que les chercheurs en sciences de l'environnement pourront utiliser. Le compilateur, le moteur d'exécution et l'environnement de modélisation que nous avons développés pour Ocelet constituent une première étape. Les étapes suivantes (qui sont hors du cadre de cette thèse) vont consister à faire évoluer cet outil de l'état de prototype à l'état d'outil opérationnel. Cela passera par une amélioration de la fiabilité, des travaux d'optimisation dans le moteur d'exécution, et l'ajout de quelques fonctions de manipulation de graphes que nous avons définies mais qui n'ont pas encore été implémentées.

Ce prototype donne déjà satisfaction dans la mesure où il nous permet d'écrire des modèles en Ocelet, de les compiler, d'exécuter des simulations et d'accéder aux résultats sous forme de cartes dynamiques. Ainsi nous avons pu mettre nos concepts de graphes d'interaction à l'épreuve de situations de modélisation très différentes. C'est une sélection de ces expériences de modélisation qui fait l'objet du chapitre suivant.

Chapitre 6

Applications de modélisation avec Ocelet

Sommaire

6.1 Introduction

Dès le début de nos réflexions pour ce travail de thèse, nous avons souhaité nous appuyer sur des cas concrets de modélisation, présentant une diversité de situations, et dans

lesquels la composante spatiale a une influence sur la dynamique d'un paysage. L'étude récurrente de ces différents systèmes a enrichi notre travail par un va et vient entre réflexion théorique sur les concepts de modélisation et confrontation de ces concepts à des situations concrètes. Ces modèles ont donc d'abord été imaginés sur le papier, en manipulant nos concepts de modélisation à travers des expériences de pensée, puis effectivement programmés et testés lorsque le langage et les outils associés sont arrivés à un niveau de finition suffisant.

Nous présentons dans ce chapitre trois des modèles que nous avons réalisés avec Ocelet afin d'illustrer certains aspects de notre approche :

1. Un modèle fictif de dissémination de pathogènes, qui est une première expérience de mise en oeuvre d'un graphe de voisinage spatial et de son usage pour calculer un phénomène de contamination progressive.
2. Un modèle de dynamique côtière, dans lequel nous avons pu simuler l'évolution de la forme du trait de côte à partir des interactions entre trois type d'entités (océan, banc de vase et chenier).
3. Un modèle d'évolution historique de l'usage des sols, dont les règles sont exprimées sous la forme d'automates. Ce modèle a par ailleurs mis à l'épreuve la capacité d'expression d'Ocelet par une introduction progressive d'hypothèses de modélisation de natures très différentes.

6.2 Modèle de dissémination de phytopathogènes entre parcelles cultivées

Certaines maladies fongiques ont la capacité de se développer sur une parcelle cultivée, puis de libérer des spores qui vont contaminer les parcelles voisines au grés des vents. C'est par exemple le cas du *sclerotinia sclerotiorum* qui s'attaque au colza, de la maladie du charbon (*ustilago scitaminea*) qui s'attaque à la canne à sucre, ou de *microcyclus ulei* qui s'attaque aux feuilles de l'hévéa. Une parcelle contaminée risque de relayer à son tour la diffusion de la maladie qui aura ainsi une capacité à s'étendre.

Nous nous sommes inspirés de ce mode de dissémination aérien de phytopathogènes pour imaginer un modèle simplifié qui reproduit une dynamique de contamination en se basant sur un graphe de voisinage. Ce graphe sert de structure à un graphe d'interaction qui porte les règles de diffusion d'un pathogène en fonction de quelques critères comme la direction et la force du vent et un coefficient de voisinage entre deux parcelles.

Le scénario de ce modèle comporte trois phases principales :

1. L'extraction du graphe de voisinage à partir d'un parcellaire stocké dans une base de données géographique.
2. La préparation de l'état initial d'une simulation.
3. La simulation elle même qui fait évoluer l'état du graphe d'interaction chaque jour en fournissant des données de direction et de force du vent. Au cours de la simulation,

un état du parcellaire est enregistré chaque année, à travers un datafacer, au format KML. Cela permet l'affichage de la dynamique avec le logiciel Google Earth. La figure 6.1 donne un exemple de résultat obtenu.

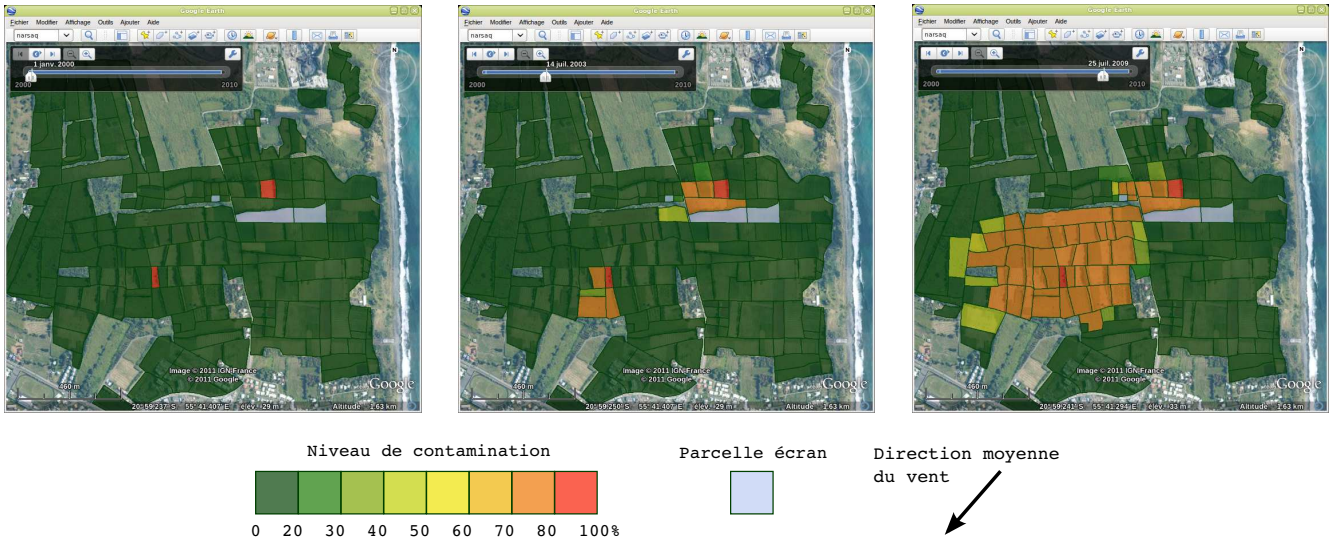


FIG. 6.1 – Déroulement d'une simulation (extraits de l'affichage avec Google Earth).

Nous exposons dans les trois sections suivantes quelques détails de construction relatifs à chacune de ces phases du modèle.

6.2.1 Accès à une base de données et obtention du voisinage

Nous disposons d'un parcellaire agricole constitué d'un ensemble de polygones géoréférencés qui sont stockés dans une table d'une base de données de type PostgreSQL/PostGIS [5]. L'écriture de ce modèle a d'ailleurs été l'occasion de développer (directement en Java) la première version d'un datafacer permettant d'accéder à une base de données de ce type et dédié à la construction de graphes de voisinage sur des parcellaires agricoles.

Ce datafacer nommé `FieldsPostgis` dispose (entre autres) d'un service pour établir la connexion à la base de données, d'un service pour extraire les centroïdes des polygones, et d'un service de calcul de voisinages.

Ces services sont déclaré dans `FieldsPostgis.oclt` de la façon suivante :

```
datafacer FieldsPostgis {
  service list[Centroid] getCentroids(text tableName);
  service connectBase(text baseUrl, text baseName, text login,
     text passwd);
  service list[Neighblink] computeNeighb(text tableName, real
     distance);
  ...
}
```

Le calcul du voisinage est effectué par une requête SQL faisant usage d'opérateurs spatiaux de PostGIS. Dans la version actuelle du datafacer le seul paramètre sur lequel le modélisateur puisse agir depuis le modèle écrit en Ocelet est la distance maximale de voisinage. Cette distance est utilisée d'abord pour décider si deux parcelles sont voisines ou non. Elle est ensuite utilisée pour calculer un coefficient de voisinage.

Nous avons en effet voulu introduire dans le modèle un coefficient qui permette une diffusion plus importante de spores si deux parcelles sont voisines par une grande partie de leur périmètres respectifs que si elles ne le sont que par un coin. Ainsi chaque arc du graphe de voisinage va porter un coefficient de voisinage dont la valeur (comprise entre 0 et 1) est fonction de la surface d'intersection entre une zone tampon autour d'une parcelle et la parcelle voisine. Sur la figure 6.2 par exemple, le coefficient de l'arc de voisinage `A` → `B` sera bien plus important que celui de l'arc de voisinage `A` → `C`.

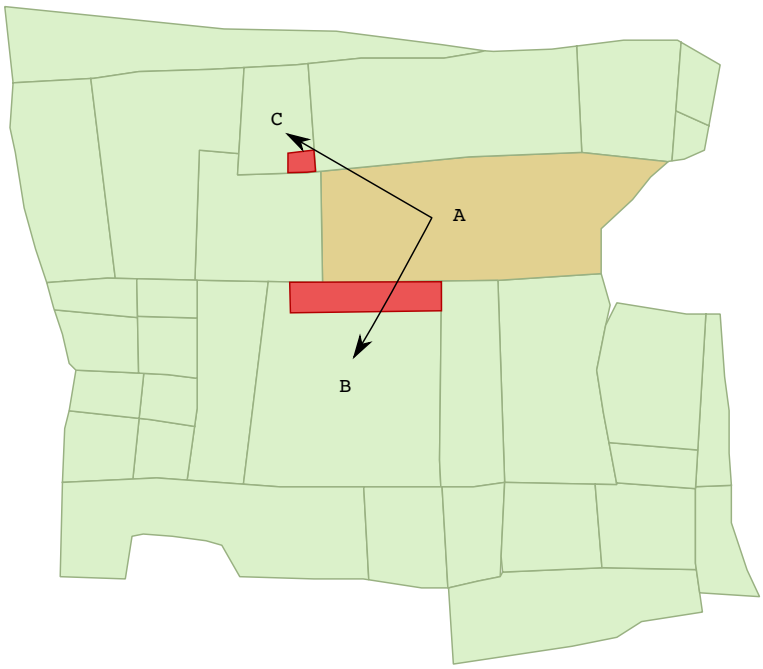


FIG. 6.2 – Calcul des voisinages et des coefficients associés à l'aide d'une zone tampon

Nous donnons ci-dessous pour le lecteur intéressé la requête SQL qu'effectue le service `computeNeighb()` du datafacer pour extraire les voisinages. Dans cette requête, `<tableName>` et `<distance>` sont remplacés par les valeurs des arguments du service :

```
SELECT S1.gid, S2.gid,
       st_area(st_intersection(st_buffer(S1.the_geom,<distance>),
                               S2.the_geom)) as inter
FROM <tableName> as S1, <tableName> as S2
WHERE (S1.gid != S2.gid) AND
      (distance(S1.the_geom,S2.the_geom) <  <distance>)
```

Dans ce modèle nous faisons deux usages du service `getCentroids()` du datafacer `FieldsPostgis` :

1. Le modèle prend en compte la direction du vent pour orienter la diffusion des spores. En disposant du centroïde des parcelles aux extrémités de chaque arc du graphe, nous pouvons calculer l'angle d'orientation de cet arc, cet angle étant dans ce cas considéré comme une approximation de la direction de voisinage entre deux parcelles. Lors de la construction du graphe de voisinage, nous attachons donc la valeur de cet angle à chacun des arcs. Ainsi le service de la relation qui calcule la dissémination des spores pourra tenir compte de l'angle de voisinage entre deux parcelles : le comparer à la direction du vent à ce pas de temps, et pondérer la diffusion en conséquence.
2. Cela nous permet de produire un affichage du graphe qui a été obtenu : nous plaçons chaque sommet de ce graphe aux coordonnées géographiques du centroïde de la parcelle qui lui correspond et nous traçons les arcs entre ces points (un exemple est donné sur la figure 6.3).

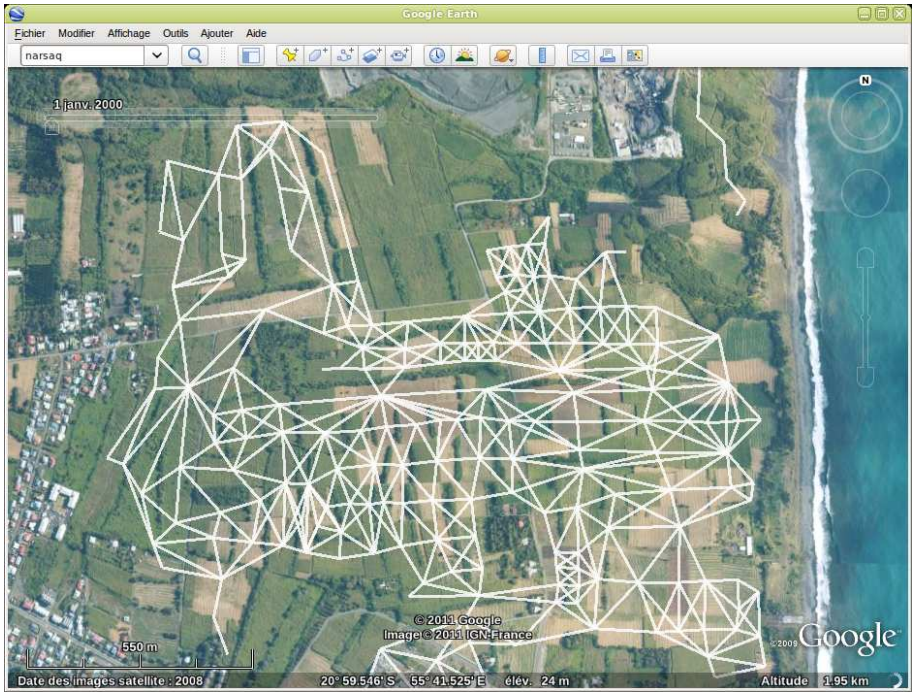


FIG. 6.3 – Exemple de graphe de voisinage obtenu.

6.2.2 Construction de l'état initial du modèle

Le modèle comporte une définition d'entité `Plot` qui correspond à une parcelle agricole et qui contient une propriété nommée `lct` reflétant son niveau de contamination.

Nous avons défini une relation de contamination qui porte la sémantique du mécanisme de diffusion de spores d'une parcelle contaminée vers une parcelle voisine. Un extrait de cette relation est donné ci-dessous :

```
relation Diffusion[source, destination] {
  property real ncoef;   // coefficient de voisinage
  property real nangle;  // angle de voisinage

  // wint et wdir representent l'intensite et la direction du vent
  service contaminate(real wint, real wdir) {
  ...
    real proba = ncoef * wint * m.cos((wdir-nangle)*3.1416/180.0)
       *2.0 ;
    real willchange = m.randomr(0.0,1.0);

    // lct represente l'etat de contamination
    curLC = destination.lct;

    // serie de regles de changement d'etat de la destination
    if (curLC == 1) {if (willchange < proba) { newLC = 2;}}
    ...
    if (curLC == 6) {if (willchange < proba) { newLC = 7;}}
    destination.lct = newLC;
  }
 ...
}
```

On notera dans cet extrait les deux propriétés `ncoef` et `nangle` qui seront donc attachées à chaque arc du graphe. Précisons aussi que ce modèle contient comme on peut le voir une part d'aléatoire dans la contamination d'une parcelle voisine. Ce modèle est fictif et les règles simplifiées de dissémination que nous avons utilisées ne sont certainement pas aussi précises que les règles que l'on pourrait appliquer avec un cas réel bien documenté.

Le service `computeNeighb()` de `FieldsPostgis` renvoie une liste de paires d'identifiants de parcelles. Chaque paire décrit un arc du graphe de voisinage. Cette même liste contient aussi le coefficient de voisinage associé à chaque paire. Une partie de la construction de l'état initial consiste simplement à instancier des entités représentant les parcelles et à les connecter à travers une instance de la relation `Diffusion`, en respectant les paires d'identifiants fournies par `computeNeighb()`. Au moment de la connexion, on initialise aussi les deux propriétés portées par chaque arc : le coefficient de voisinage et l'angle de voisinage déduit de la position relative des centroïdes :

```
  ng = Diffusion[Plot, Plot];
  ...
  nlinkslist = fpg.computeNeighb("brapan_rgr92", 10.0);
  ...
  while (nlinkslist.hasNextRecord()) {
     nlink = nlinkslist.getNextRecord();
     ...
     ng.connect(plotsrc,plodst){ncoef = nlink.getCoef(); nangle =
```

```
        arcangle;};
  }
```

Nous avons marqué l'état de certaines parcelles directement dans la base de données pour constituer un état initial. Quelques parcelles sont contaminées dès l'état initial. D'autres sont mise à l'état de parcelles *écran*, qui sont par exemple plantées avec des variétés ou des espèces qui sont défavorables au développement des champignons pathogènes. Pour constituer l'état initial du système nous initialisons donc simplement chaque instance d'entité `Plot` avec l'état que nous donne le datafacer.

6.2.3 Simulation de la dynamique de contamination

Nous disposons d'une série journalière d'intensités et de directions du vent. Ces données sont stockées dans un fichier au format *Comma Separated Values* (`csv`). Nous faisons usage d'un datafacer nommé `Csvfile` qui est dédié à la lecture de fichiers de ce type.

Ce modèle ne contient qu'un seul scénario nommé `Contamination` qui se charge d'effectuer l'une après l'autre les trois phases que nous avons évoqué.

Dans la troisième phase, dédiée à la simulation elle même, ce scénario est relativement simple. Il consiste principalement à lire les données journalières du vent et à appeler le service `contaminate()` de la relation `Diffusion`. Ce seul appel de service va effectuer des interactions simultanées sur l'ensemble des arcs du graphe et donc faire évoluer l'état des parcelles. Nous donnons ci-dessous un extrait de ce scénario :

```
scenario Contamination {
  ...

  for (year in 2000..2010)
    {
      ...
      for (day in 1..365) {
       // Lecture des donnees du vent
       if (w_csv.hasNextRecord()) {
         arecord  = w_csv.getNextRecord();
         wi = arecord.getColumnAsReal(1);
         wd = arecord.getColumnAsReal(3);
       }
      ng.contaminate(allvc, wi, wd);
     }
   }
```

Ne sont présentés ici que les aspects du modèle relatifs au mécanisme de contamination. Le scenario comporte aussi des instructions permettant d'exporter l'état du parcellaire une fois par an au format KML.

6.2.4 Discussion

Ce modèle de dissémination n'a été développé que pour illustrer la capacité d'Ocelet à simuler ce type de processus à l'aide de graphes de voisinages. Bien entendu si l'on réalise un modèle plus réaliste on peut adapter la façon de construire le graphe de voisinage, on peut spécifier des règles de contamination différentes et on peut ajouter d'autres relations pour introduire davantage de mécanismes comme par exemple les plantations, la croissance et la récolte ou encore l'apport de phytopathogènes par des animaux.

Le datafacer `FieldsPostgis` que nous avons développé à l'occasion de la conception de ce modèle est pour l'instant dépendant d'une certaine forme de structure de la table de base de données contenant un parcellaire agricole. Nous avons par exemple supposé que l'attribut contenant la géométrie des parcelles se nomme `the_geom`. Même si ce nom est extrêmement courant parce qu'il est utilisé par défaut par PostGIS lors de l'importation de données, rien ne garantit qu'il est toujours utilisé. Ce sont là des questions d'ingénierie que nous pourrons améliorer pour rendre ce datafacer plus générique.

On remarquera que l'on fait appel à un petit groupe de datafacers différents pour lire des données géographiques dans une base de données, lire des valeurs d'intensité et de direction du vent dans un fichier `csv`, et exporter des résultats au format KML pour profiter de la capacité de Google Earth à afficher des représentations cartographiques dynamiques. La dynamique elle même est calculée sur un graphe et non sur la géométrie des parcelles. Tous les calculs sont effectués en coordonnées métriques dans un système projeté, mais le datafacer d'export vers le format KML s'occupe de convertir ces coordonnées en latitudes et longitudes dans le système de coordonnées non projeté qu'utilise Google Earth. Cette conversion est quasiment transparente pour le modélisateur.

Cette combinaison de formats et leur couplage avec la manipulation de graphes d'interaction est une illustration concrète de la souplesse que nous avons voulu donner à Ocelet dans la manipulation de l'information spatiale.

6.3 Dynamique côtière d'un écosystème de mangrove

Les côtes des Guyanes, bordées de mangrove sur une longueur de 1600 km, de l'embouchure de l'Amazone au delta de l'Orenoque, reçoivent près de 20% de la quantité de matières en suspension déversées annuellement par l'Amazone dans l'océan atlantique [90]. Une partie de cette énorme quantité de matières en suspension est déviée par le courant Nord Brésilien vers le Nord-Ouest. Les processus hydro-sédimentaires en action sont complexes à étudier en raison de leurs natures (actions de la houle océanique et des courants sur le transport, le dépôt ou la remise en suspension de sédiments fins et cohésifs) et des échelles spatiales et temporelles auxquelles il faut les appréhender (du banc de vase à la région littorale ; de l'année à plusieurs décennies) pour prédire les futurs changements côtiers [12]. Sous l'action des houles [76], ces matières en suspension s'agglomèrent à la côte et forment des bancs de vase dont la longueur varie entre 10 à 40 km le long des côtes pour 4 à 5 km en direction de l'océan dans leur partie exondée à marée basse. Les vitesses

de migration vers le Venezuela de ces plateformes vaseuses varient entre 300m et 3km par an [61, 65]. Entre chaque banc de vase (période d'accrétion vaseuse), une phase d'érosion remet en suspension la vase agglomérée et déracine la mangrove qui poussait dessus.

Les travaux visant à décrire le fonctionnement et la dynamique de ces interactions océan-vase-mangrove mettent en évidence un système extrêmement complexe dans lequel des processus sédimentaires, écologiques, morphodynamiques, biogéochimiques et océanographiques se trouvent intimement liés à différentes échelles spatiales et temporelles [15, 62, 125].

Il n'y a pas encore de modèle capable de simuler à plusieurs échelles spatiales d'une part 1) les variations géomorphologiques du littoral guyanais et d'autre part 2) les influences de cette instabilité côtière sur le fonctionnement et le maintien de l'écosystème de mangrove, principal écosystème côtier de la région [124].

Le modèle que nous présentons ici est une tentative de simulation de la dynamique côtière de la mangrove sur une partie de la côte Guyanaise d'environ 50 km, située entre Cayenne et Kourou entre 1986 et 2008. Nous nous servons de séries temporelles d'images SPOT acquises sur le littoral guyanais pour valider nos simulations.

6.3.1 Les éléments du modèle de dynamique côtière

Ce modèle prend en compte quatre éléments (figure 6.4) :

Le cordon sableux (ou chenier) est dans notre modèle considéré comme la ligne de côte de référence à partir de laquelle on mesure l'extension de la mangrove vers la mer. Nous considérons ici le chenier comme une ligne stable qui ne peut être érodée.

Le banc de vase représente dans notre modèle une zone de vase agglomérée à la côte, qu'elle soit émergée à marée basse mais non encore colonisée par la mangrove, ou en permanence immergée (zone d'amortissement des houles). Le banc de vase se déplace dans le sens du courant dominant, par l'effet de phénomènes d'érosion dans sa partie sud-est et de déposition dans sa partie nord-ouest.

La zone de mangrove correspond à une zone de vase où un écosystème de mangrove se développe. La limite de cette zone représente ce qu'on appellera 'trait de côte'. Celui-ci peut rejoindre la ligne de chenier durant les phases d'érosion intense. Ce sont les variations de ce trait de côte, autrement dit la géomorphologie des bancs de vases colonisés par la mangrove que nous cherchons à simuler.

L'océan est une entité qui comprend à la fois un signal de houle et un signal, pour l'instant constant, de courant parallèle à la côte de référence.

6.3.2 Les données et leur intégration dans le modèle

Nous disposons d'une série d'images de télédétection couvrant la zone entre Cayenne et Kourou pour les années 1986,1988,1990,1995,2001,2002,2003,2006 et 2010. Les images sont acquises à différentes hauteurs d'eau. Comme la topographie des bancs de vase n'est

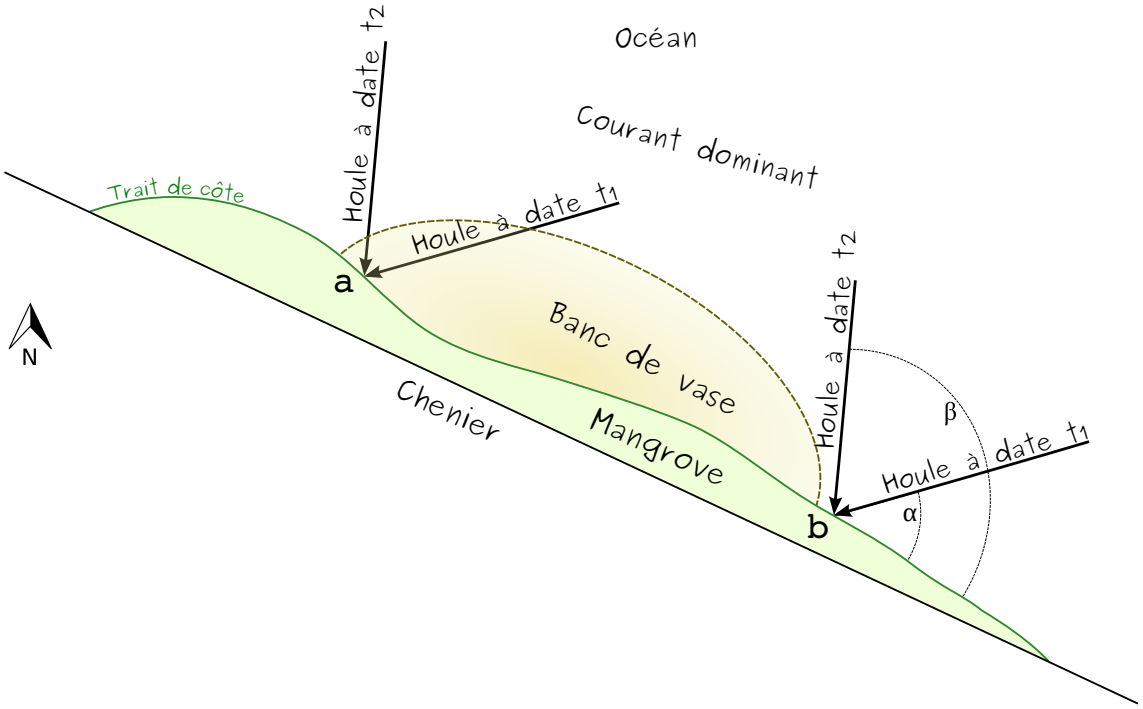


FIG. 6.4 – Principaux éléments du modèle de dynamique côtière d'un écosystème de mangrove.

pas connue et varie, il est encore impossible de donner et suivre d'une date à l'autre une surface de vase à une hauteur d'eau fixe. Seule l'apparition de la mangrove donne une information observable dans toutes les images.

Nous avons utilisé la première image pour saisir deux formes linéaires géoréférencées : celle du chenier et celle du trait de côte à la date de cette image. Ces formes sont stockées dans un fichier au format shapefile. Le modèle accède à ces formes linéaires à travers un datafacer `Shapefile`. Ce datafacer permet de lire le contenu de fichiers de ce format, mais il propose aussi des services pour manipuler des formes géométriques de type point, ligne ou polygone. Ces services nous donnent en particulier accès aux coordonnées de chaque point d'une forme linéaire, dont nous avons besoin pour faire évoluer la forme du trait de côte.

Les images de télédétection ont aussi servi à observer les différences entre le trait de côte visible et celui qui est simulé par le modèle. Ces observations ont été utiles pour calibrer certains paramètres du modèle.

Le signal de houle océanique est obtenu de la base de données simulées ERA40 mise à diposition par le ECMWF[23]. C'est un signal simulé à raison de 4 données par jour sur un maillage spatial de 1.5 degrés depuis 1957. Pour notre étude, nous nous servons de l'énergie de la houle et de sa direction. Ces données sont stockées dans un fichier de type texte au format csv et nous y accédons à l'aide d'un datafacer Csvfile.

[23](European Centre for Medium-Range Weather Forecasts.
http ://www.ecmwf.int/products/data/archive/descriptions/e4/index.html)

6.3.3 Principes de simulation

Le courant dominant apporte des sédiments en suspension qui se déposent contre le chenier et/ou contre le banc de vase, dans les zones protégées par un banc de vase de l'action re-mobilisatrice de la houle. Plus l'énergie de la houle est élevée, plus son action érosive est importante. Si la houle vient du Sud-Est, son action va se combiner avec celle du courant pour 'accélérer' la migration du banc de vase vers le Nord-Ouest. Au contraire, si la houle vient du Nord-Ouest et est de forte énergie (période de Novembre et tempêtes océaniques), la vitesse de transport des sédiments vers le Nord-Ouest pourrait diminuer avec un impact sur la physionomie de l'avant banc (partie Nord-Ouest) qui augmente. Dans ce scénario, la taille, la position et le volume du banc de vase, notamment sa partie immergée seraient des données fondamentales. Malheureusement, il faut bien voir qu'aucune technique ne permet encore de chiffrer ces données. L'état initial et la position du banc de vase se fait donc par interprétation visuelle des images et une certaine dose d'apprentissage acquise dans l'observation des images de télédétection et les expérimentations de terrain.

Sur le schéma de la figure 6.4, la houle qui frappe le point `b` est en contact direct avec le trait de côte et cela va engendrer une érosion rapide, alors qu'au point `a`, le banc de vase atténue ses effets. Cette atténuation sera d'autant plus forte que l'angle d'attaque de la houle l'amènera à parcourir une grande surface de banc de vase. Sur la même figure, la houle arrivant au point `a` avec un angle β (à la date t2) aura une distance moins grande à parcourir sur le banc de vase pour attaquer le trait de côte que lorsqu'elle arrive avec un angle α (à la date t1). La houle accentue aussi, de manière irrégulière, les effets d'érosion et de re-déposition de matières sur le banc de vase, ce qui a une influence sur la célérité du glissement de ce banc de vase le long de la côte.

6.3.4 Initialisation et simulation

Nous avons défini une entité pour chacun des éléments que nous venons de décrire : `Chenier`, `Coastline` qui représente la mangrove, `Mudbank` qui représente le banc de vase et `Ocean`.

On voit dans l'extrait du modèle ci-dessous que les entités `Chenier` et `Coastline` possèdent toutes deux une propriété de type `ShpRecord` qui contient une forme géométrique (linéaire dans le cas présent). Ce type `ShpRecord` est défini dans le datafacer `Shapefile` et il correspond en fait à un enregistrement d'un fichier shapefile.

Nous avons aussi inclu dans cet extrait le code du service `getLine()` de `Coastline` qui montre comment l'on peut obtenir une liste de `Positions` (c'est à dire des points avec leur coordonnées) à partir d'une forme de type `ShpRecord`.

```
// L'entite Chenier ne changera pas durant la simulation
entity Chenier {
  property ShpRecord record;
 }

// L'entite Coastline va se deformer par l'effet des interactions
```

```
// avec l'entite Ocean.
entity Coastline {
  property ShpRecord record;
  . . .
  service list[Position] getLine(int numLine) {
    pl = record.getPolyline();
    return pl.getLine(numLine);
  }
  . . .
 }
```

Préparation de l'état initial

Les entités `Chenier` et `Coastline` sont initialisées au début du scénario avec les coordonnées de leurs formes linéaires respectives lues dans un fichier shapefile ; le même fichier contient les deux formes dans des enregistrements successifs. La phase d'initialisation contient le code suivant destiné à construire des instances de `Chenier` et `Coastline` avec leur état initial :

```
scenario Mangrove {
  text shp_name="data/macouria_sim1chenier+1986.shp";
  shp = ShapeFile{};
  shp.setFileName(shp_name);
  shp.setCrsEPSG("EPSG:32622");
  chenier = Chenier{record = shp.getNextRecord();};
  coastline = Coastline{record = shp.getNextRecord();};
  . . .
  }
```

L'entité `Mudbank` n'est pas initialisée avec des données mais simplement avec la position d'un point de référence représentant la position initiale d'un banc de vase. Nous n'avons pas de donnée concernant la forme précise des bancs de vase, ceux-ci étant immergés pour leur plus grande partie, ils sont particulièrement difficile à observer et à mesurer. Le banc de vase utilisé dans les calculs est une demi-ellipse tracée autour de ce point de référence. Le scénario va faire glisser ce point de référence le long du chenier à une vitesse variable qui tient compte du courant et de la houle, la demi-ellipse suivra donc automatiquement ce déplacement.

L'initialisation de l'entité `Ocean` consiste à préparer la lecture séquentielle des données de houle. Cela est effectué de la manière suivante :

```
  ocean = Ocean{};
  ocean.w_csv = Csvfile{} ;
  ocean.w_csv.setFileName("data/Wave_intensity_direction_1986
     -2010.csv");
```

L'intensité et la direction du courant dominant sont des listes de 12 valeurs moyennes, c'est à dire une valeur par mois. Ces valeurs sont reprises à l'identique d'une année sur l'autre, et le fait d'avoir une valeur différente par mois permet de rendre compte de variations saisonnières du courant. Ces douzes valeurs ont été pour l'instant incluses dans la définition même de l'entité `Ocean`, et il est probable que dans une version ultérieure nous les lirons dans un fichier.

Déroulement d'une simulation

Nous avons vu que l'océan est le véritable moteur de la dynamique du trait de côte et des bancs de vases. L'interaction principale à prendre en compte dans ce modèle fait intervenir l'océan, le trait de côte et le banc de vase. C'est donc en théorie une relation à trois rôles que nous devrions écrire avec Ocelet. A l'heure de la réalisation de ce modèle, la gestion des hypergraphes n'étant pas encore disponible dans le moteur d'exécution d'Ocelet, cette relation à trois rôles a dû être simulée à l'aide d'une relation à deux rôles qui est définie de la manière suivante (extrait) :

```
relation Forcing [coast, waves]{
  service modify_coast(int month, real x_0, real y_0) {
    real c_i = waves.getCurrentIntensity(month); // current
       intensity
    real c_d = waves.getCurrentDirection(month); // current
       direction
    . . .
    newcoord = list[Position];
    coord = coast.getLine(j); // obtain de points coordinates of
       the coast
    newcoord.add(coord.get(0));
    for (i in 1 .. (coord.size()-2)) {
    . . .
// calcul du deplacement de chaque point du trait de cote en
// fonction du courant, de la houle, et de l'attenuation due
// au banc de vase
    . . .
    }
    . . .
    newcoord.add(coord.get(coord.size()-1));
    newpl.addLine(newcoord);
    coast.setPolyline(newpl); // update modifications in 'coast'
  }
}
```

Les deux rôles principaux entre l'océan et le trait de côte sont effectivement définis commes des rôles dans la relation. Par contre le rôle joué par le banc de vase est introduit dans le service `modify_coast` sous la forme des arguments `x_0` et `y_0`. Ces variables sont les coordonnées du point de référence permettant de localiser le banc de vase.

Les portions du service que nous avons choisi de présenter dans cet extrait illustrent le principe de son fonctionnement :

- lecture de l'intensité et de la direction du courant pour le mois en cours ;
- création d'une nouvelle ligne de côte dont le premier point est fixé ;
- calcul (non détaillé dans cet extrait[24]) de chacun des points de cette nouvelle ligne de côte à partir des points de ligne actuelle, du courant, de la houle (que l'on obtient du rôle `waves`), et de l'atténuation dûe au banc de vase.
- remplacement des coordonnées de l'ancien trait de côte par celles du nouveau trait de côte.

Le graphe d'interaction mis en oeuvre dans le scénario pour cette relation est minimal puisqu'il ne contient que deux sommets reliés par un arc :

```
scenario Mangrove {
  . . .
  // relate coastline and ocean
  f = Forcing[Coastline, Ocean];
  f.connect(coastline, ocean);
  . . .
}
```

Deux autres relations sont à prendre en compte que ne détaillons pas ici :

1. l'interaction entre le trait de côte et le chenier, en particulier pour éviter que l'érosion du trait de côte ne dépasse la limite que constitue le chenier ;
2. l'interaction entre l'océan et le banc de vase, pour déplacer le point de référence de l'ellipse qui le représente.

La simulation va faire évoluer le modèle en calculant les interactions chaque jour, et exporter un état du trait de côte et du banc de vase chaque mois dans un fichier au format kml (pour affichage dans Google Earth) :

```
scenario Mangrove {
  . . .

  for (year in 1987..2010) {
    for (doy in 1..366) {
      month = 1 + doy/31;
      . . .
      f.modify_coast(month, x0, y0);
      . . .
    }
    kml.addStyledRecord("Mobile mudbank "+year, ""+year+"-01-01",
                        ""+year+"-12-31",onerec,"0");
```

[24] ce calcul fait principalement appel à des fonctions de trigonométrie qui ne sont pas spécifiques à Ocelet. Les équations détaillées feront l'objet d'une publication (en cours de rédaction) dédiée à ce modèle.

```
    kml.addStyledRecord("Coastline 2011", "2011-01-01",
       "2011-12-31",
                          coastline.record, "6");
  }
  print ("Saving the kml file in output ....");
  kml.saveAsKml("output/Macouria1986-2010.kml");
  . . .
}
```

6.3.5 Résultats et discussion

Ce premier test montre que la tendance générale de l'évolution géomorphologique est assez bien retrouvée dans la région de Cayenne-Kourou. C'est un résultat très intéressant car il visualise (figure 6.5) l'hypothèse que l'équipe guyanaise (ULCO-CEREGE-AMAP) faisait sur la prépondérance du forçage 'houle' dans l'explication des variations géomorphologiques du littoral guyanais. Il va s'agir maintenant de poursuivre l'analyse en intégrant un plus grand nombre d'images satellitales et en travaillant sur deux autres régions notamment au Nord de la Guyane où l'on observe un changement d'orientation de la côte (les bancs de vase devraient s'allonger) et également une modification importante de l'embouchure des rivières Sinnamary et Mana.

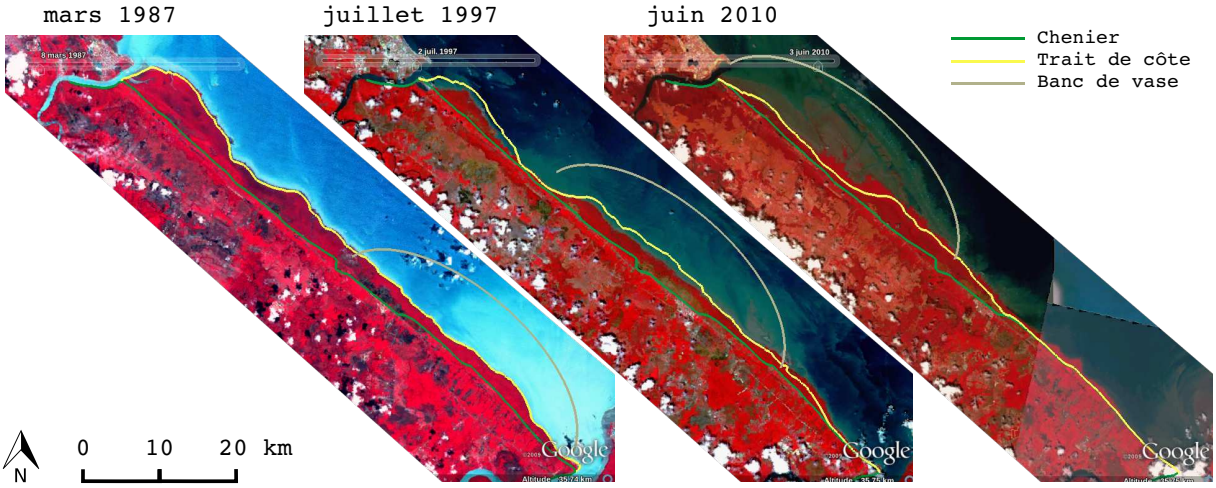


FIG. 6.5 – Résultats d'une simulation illustrant l'état du trait de côte et la position du banc de vase pour les années 1987, 1997 et 2010. Sur ces compositions colorées, la végétation apparait en rouge.

Parallèlement, il s'agira de décrire et d'intégrer dans le modèle, d'une part, les processus de colonisation d'un banc de vase exondé en fonction des cycles de marée et si possible de sa topographie [55, 125] et, d'autre part, les processus de dynamique forestière en utilisant par exemple des modèles déjà développés [19]. Dans ce cadre, il pourrait y avoir également un autre pilotage par observations spatiales, cette fois-ci de résolution métrique, pour obtenir des validations de l'état structural de la forêt de mangrove.

La forme du banc de vase pourrait aussi être décrite avec davantage de précision. Nous lui avons donné une forme décrite par l'équation d'une ellipse, mais bien entendu les bancs de vase ont une forme plus complexe. On sait par exemple qu'il ont tendance à avoir une densité de vase plus importante à l'arrière de leur déplacement (côté sud-est dans notre exemple) qu'à l'avant (nord-ouest). Une relation réellement définie avec trois rôles nous permettrait de prendre en compte une entité banc de vase un peu plus réaliste au lieu de travailler sur une demi-ellipse homogène.

Cet exemple nous a permis d'expérimenter Ocelet dans une situation où l'on cherche à déplacer et déformer des objets ayant une structure spatiale. Ce résultat est obtenu avec une bonne séparation des mécanismes d'interaction, cela permet de ne réfléchir que sur un processus d'interaction à la fois, et devrait faciliter l'évolution et l'amélioration du modèle.

La solution que nous avons mis en œuvre n'est pas la seule possible. Nous avons choisi de traiter les formes linéaires présentes dans le modèle (trait de côte et chenier) comme des listes de points. Nous pourrions mettre ces points non pas dans une liste mais dans un graphe, chaque point étant connecté à ceux avec lesquels il doit intéragir. Cela nous amènerait à changer dynamiquement la topologie du graphe pour tenir compte d'anciens voisinages qui disparaissent et de nouveaux qui se forment, au fur et à mesure du déplacement du banc de vase. C'est là une autre façon d'aborder ce modèle qui mérite d'être expérimentée.

6.4 Evolution de l'usage du sol dans le sud de l'Inde

Le district du Coorg (Kodagu) dans l'état du Karnataka, dans le sud de l'Inde, est caractérisé par un terrain vallonné, des pentes douces, un régime climatique propice à la culture du café, et des zones de bas-fonds qui conviennent à la riziculture. Durant ces dernières décennies, la culture du café ne s'est plus cantonnée à sa zone traditionnelle (que l'on appelle "coffee belt") et a commencé à grignoter sur la forêt. La forêt naturelle a d'abord été éclaircie afin de permettre au café de profiter de l'ombre de grands arbres émergents. Ceux-ci ont graduellement été remplacés par des arbres à croissance rapide tel le Grevillea robusta (nommé Silver Oak en anglais) pour fournir l'ombre nécessaire au café. Les bas-fonds cultivés essentiellement en riz ont également fait l'objet d'évolutions, parfois cycliques avec des périodes de mise en jachère, parfois plus durables avec une conversion vers d'autres cultures.

Nous nous sommes appuyés sur le travail de J. Alet qui a caractérisé et modélisé ces évolutions, dans une étude récente, sur une zone proche du village de Kottoli [10]. Elle a décrit différentes hypothèses d'évolution sous la forme d'automates à états finis, dans lesquels chaque état correspond à un type d'occupation du sol et chaque transition entre états est associée à une probabilité de réalisation. D'autres règles ont été progressivement ajoutées comme le changement d'automate à certains pas de temps, et des critères de voisinage d'occupation du sol, ou d'appartenance à une catégorie d'exploitation. L'implé-

mentation des différentes hypothèses et simulations a été réalisée sur la plateforme Dypal [71].

Nous avons voulu étudier comment reproduire certains aspects de ce modèle à l'aide d'Ocelet, voir les questions méthodologiques que cela soulève, et tenter d'y apporter des réponses.

Différentes hypothèses de modélisation ont été proposées par J. Alet, et nous en avons retenu quatre :

1. Un même automate est utilisé sur toute la zone et sur toute la période de simulation, avec des probabilités de transition fixées à l'avance.
2. L'automate change au cours du temps, pour rendre compte de certains évènements qui ont provoqué des changements dans le comportement des fermiers.
3. Les fermiers sont regroupés en quatre catégories selon leurs pratiques culturales habituelles. A chaque catégorie correspondent des automates différents. Les règles de transition des parcelles sont donc dépendantes du temps et du type de pratique agricole.
4. Certaines transitions sont dépendantes de l'état de parcelles voisines.

6.4.1 Eléments communs du modèle

Il s'agit de simuler, dans les quatre hypothèses, l'évolution de l'occupation du sol entre 1950 et 2010, avec un pas de temps de 5 ans.

Nous disposons du parcellaire couvrant la zone d'étude, sous la forme d'un fichier au format shapefile. Les parcelles de ce fichier possèdent un attribut d'occupation du sol que nous utilisons pour établir l'état initial du modèle (en 1950). Nous pouvons modéliser les parcelles par une entité `Plot` :

```
entity Plot {
        property ShpRecord record;
        property int id;
        property int lct;   // land cover type
 }
```

Les données sont importées dans le modèle en faisant usage du même datafacer `Shapefile` que dans les modèles précédents. La propriété `lct` est initialisée avec le code d'occupation du sol présent dans les attributs du fichier. Ce code correspond à l'un des neufs états définis par J. Alet :

1	Cultivated paddy :	Culture de riz
2	Converted paddy :	Culture de riz reconvertie en un autre type de culture
3	Fallow paddy :	Parcelle (habituellement de riz) laissée en jachère
4	Private forest :	Forêt naturelle, attribuée à des fermiers, pas encore exploitée
5	Jungle coffee :	Culture de café sous couvert de forêt naturelle
6	Open jungle coffee :	Culture de café sous couvert de forêt éclaircie
7	Mix SO-jungle coffee :	Culture de café sous couvert d'une forêt mixte, partiellement composée de Grevillea
8	SO coffee :	Café cultivé sous couvert de Grevillea
9	Devarakadu	Petite zone de forêt sacrée, non cultivée

6.4.2 Première hypothèse : modèle comportant un seul automate

Nous avons besoin de décrire le contenu d'un automate, puis d'appliquer les règles de cet automate sur l'ensemble des parcelles à chaque pas de temps.

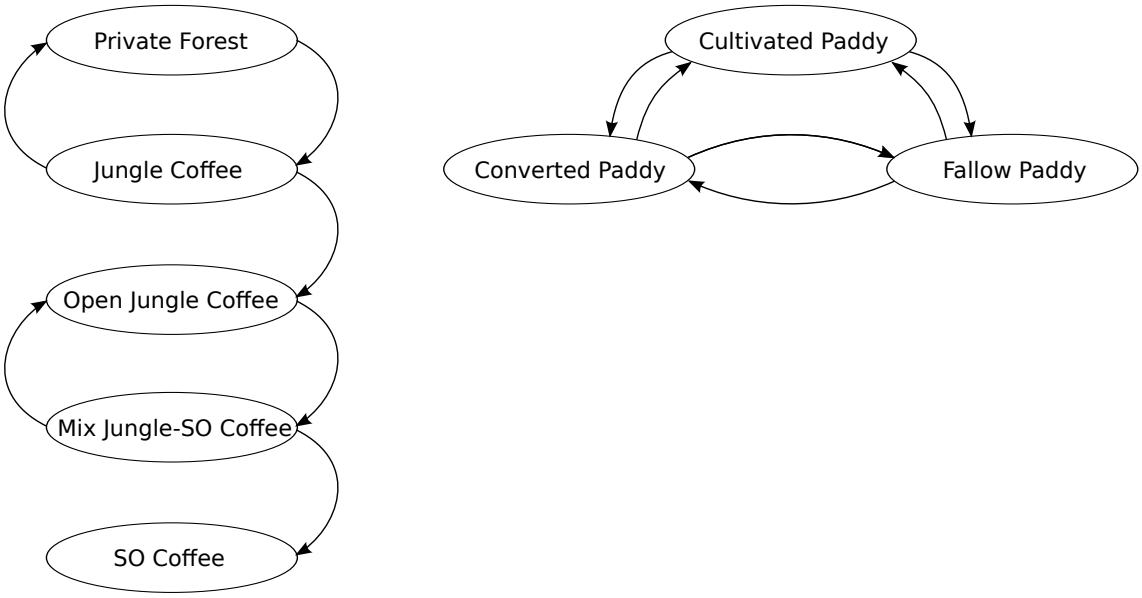


FIG. 6.6 – Diagramme des états et transitions possibles de la première hypothèse (d'après J. Alet, 2011 [10]).

Dans le cas de cette première hypothèse, chaque transition peut être décrite à l'aide d'une structure contenant l'état atteint après la transition (`toState`), et la probabilité `p` de réalisation de cette transition :

```
structure Transition {
  property int toState;
  property real p;
}
```

Un état peut également être décrit comme une structure, dans laquelle on place la liste des transitions possibles à partir de cet état. On voit que dans l'automate à construire, il n'y a jamais plus de deux transitions à partir d'un même état :

```
structure State {
  property Transition t1;
  property Transition t2;
}
```

L'automate complet prend alors la forme d'une liste contenant les neufs états, chacun portant les transitions dont il est l'origine. L'automate à construire est illustré sur la figure 6.6. La phase d'initialisation du scénario dans laquelle on construit cet automate est la suivante (extrait) :

```
scenario Kottoli {
  . . .

  // Instantiation d'un StudyArea avec son automate
  ls = list[State];

  // 1: Cultivated Paddy
  nstate1 = State{nbTrans=2; t1=Transition{toState=2; p=0.05;};
                  t2=Transition{toState=3; p=0.02;};};
  ls.add(nstate1);
  . . .
  // 4: Private forest
  nstate4 = State{nbTrans=1; t1=Transition{toState=5; p=0.14;};};
  ls.add(nstate4);
  . . .
  // 9: Devarakadu
  nstate9 = State{nbTrans=0;};
  ls.add(nstate9);
  . . .

  studyarea = StudyArea{automata=ls;};
}
```

On voit sur la dernière ligne de cet extrait l'instanciation d'une entité `StudyArea` et son initialisation avec l'automate que l'on vient de spécifier. Cette entité est définie très simplement comme suit :

```
entity StudyArea {
  property list[State] automata;
}
```

Une relation `IsPartOf` modélise l'appartenance de parcelles à une communauté ayant un comportement donné, c'est à dire sur lesquelles on applique un même automate. Nous construisons un graphe de cette relation, structuré en étoile : toutes les parcelles de la zone d'étude sont connectées à une instance de `StudyArea`.

Un seul appel au service `changeLC` (ci-dessous) de la relation à chaque pas de temps permet alors de décider, à travers les règles de l'automate, du changement d'état de chacune des parcelles.

```
relation IsPartOf [member,community]{
  service changeLC() {
    m = Math{};
    State curState = community.automata.get(member.lct-1);
    if ((curState.nbTrans > 0) && (m.randomr(0.0,1.0) < curState.
       t1.p)) {
        member.lct = curState.t1.toState;
     }
    if ((curState.nbTrans > 1) && (m.randomr(0.0,1.0) < curState.
       t2.p)) {
        member.lct = curState.t2.toState;
     }
  }
  . . .
}
```

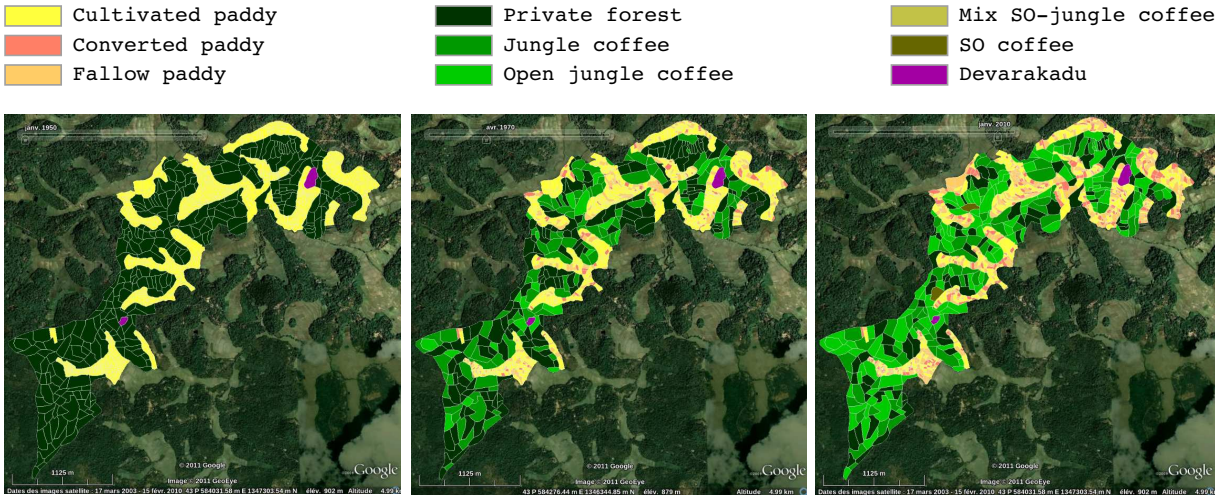


FIG. 6.7 – Exemple de simulation pour l'hypothèse 1

6.4.3 Seconde hypothèse : évolution de l'automate au cours du temps

On souhaite prendre en compte le fait qu'entre 1950 et aujourd'hui, les règlementations ont évolué et les comportements des fermiers également. On part d'un automate restreint en 1950 et celui-ci s'enrichit à la faveur des certains évènements, par l'ajout de nouvelles transitions, jusqu'à obtenir l'automate complet que nous avons mis en oeuvre dans l'hypothèse précédente.

Nous avons ajouté une propriété (nommée `tstep`) à la description d'une transition permettant d'indiquer à partir de quel pas de temps cette transition doit être prise en compte par le modèle. La partie initialisation du scénario a été adaptée pour ajouter cette contrainte à chaque transition de l'automate.

```
structure Transition {
        property int toState;
        property int tstep;
        property real p;
}

. . .

scenario Kottoli {
  . . .

  ls = list[State];

  // 1: Cultivated Paddy
  nstate1 = State{nbTrans=2; t1=Transition{toState=2; tstep=9; p
     =0.1;};
                t2=Transition{toState=3; tstep=10; p=0.8;};};
  ls.add(nstate1);
  . . .
}
```

La relation a dû elle aussi subir une légère adaptation : nous passons désormais un argument indiquant le pas de temps courant au service `changeLC`, et nous en faisons usage pour n'activer les transitions que lorsque la contrainte temporelle est vérifiée :

```
relation IsPartOf [member,community] {
  service changeLC(int timestep) {
    . . .
    if ((curState.nbTrans > 0) && (timestep > curState.t1.tstep)
       && (m.randomr(0.0,1.0) < curState.t1.p)) {
      member.lct = curState.t1.toState;
     }
    if ((curState.nbTrans > 1) && (timestep > curState.t2.tstep)
```

```
        && (m.randomr(0.0,1.0) < curState.t2.p)) {
      member.lct = curState.t2.toState;
     }
  }
}
```

6.4.4 Troisième hypothèse : plusieurs automates, liés à des groupes de fermiers

Les fermiers n'ont pas tous la même façon de gérer l'occupation du sol sur leurs parcelles. L'automate utilisé dans les deux hypothèses précédentes représente le comportement considéré comme le plus commun, mais on peut tenter d'affiner le modèle en rendant compte d'une certaine diversité dans les stratégies des fermiers. J. Alet a proposé une classification en quatre groupes de fermiers, chaque groupe représentant une façon de gérer les parcelles, et elle a associé un automate différent à chacun de ces groupes.

Il nous a donc fallu adapter le modèle de façon à ce que chaque parcelle évolue selon l'un de ces quatre automates, sachant que le groupe d'appartenance est un attribut disponible dans le parcellaire.

Au lieu de ne créer qu'une instance de l'entité `StudyArea`, nous en avons créé quatre : une pour chaque groupe de fermiers. Chacune de ces quatre instances de `StudyArea` a été initialisée avec l'automate qui lui correspond dans la phase d'initialisation du scénario.

Au lieu de construire un graphe en étoile, nous avons construit un graphe comportant quatre composantes connexes, chacune reliant une instance de `StudyArea` à un ensemble de parcelles associées à un même groupe de fermiers.

```
scenario Kottoli {
  . . .
 // Construction de quatre groupes de fermiers
  group1 = StudyArea{automata=ls_1;};
  group2 = StudyArea{automata=ls_2;};
  group3 = StudyArea{automata=ls_3;};
  group4 = StudyArea{automata=ls_4;};
  . . .
  while (shp.hasNextRecord()) {
    onerec = shp.getNextRecord();
    aplot = Plot {record = onerec; lct = onerec.getIntAttribute("
       LC");
                          id = onerec.getIntAttribute("ID");};

 // Connexion de chaque parcelle a un groupe selon la valeur de
    FARMGROUP
    gnum = onerec.getIntAttribute("FARMGROUP");
    if (gnum == 1) { r.connect(aplot, group1); }
    else {
    if (gnum == 2) { r.connect(aplot, group2); }
```

```
    else {
    if (gnum == 3) { r.connect(aplot, group3); }
    else {
    if (gnum == 4) { r.connect(aplot, group4); }
    }}}
  }
  . . .
}
```

Nous n'avons pas eu besoin de modifier davantage le modèle : la relation `IsPartOf` reste la même, ainsi que la partie du scénario qui effectue la simulation.

Il est intéressant de noter que nous avons adapté le comportement du modèle à cette troisième hypothèse, non pas en modifiant la façon dont se font les calculs de l'occupation du sol, mais simplement en changeant la structure du graphe d'interaction.

6.4.5 Quatrième hypothèse : prise en compte d'un voisinage spatial

Les parcelles de forêts disposent de meilleures infrastructures de transport que les zones cultivées en riz, permettant la circulation de tracteurs agricoles. De ce fait, la conversion de riz en un autre type de culture est toujours effectuée sur des parcelles qui peuvent bénéficier de la proximité de ces infrastructures. Autrement dit, les parcelles qui passent de l'état 1 (Culture de riz) à l'état 2 (Riz reconverti en un autre type de culture) sont toujours au voisinage immédiat soit d'une parcelle de forêt, soit de parcelles déjà converties.

Pour rendre compte de cette contrainte, il faut conditionner la règle de transition 1 $\rightarrow$ 2 de l'automate à l'adjacence d'au moins une parcelle se trouvant dans l'un des états 2,4,5,6,7,8 ou 9.

Nous avons défini une relation (nommée `NextToRice`) permettant de tester cette condition. Le graphe construit pour cette relation est simplement un graphe de voisinage (figure 6.8) que nous avons produit de la même façon que celui présenté dans le premier modèle de ce chapitre (6.2). Cette relation ne contient qu'un seul service qui marque (à travers une propriété booléenne `sel` que nous avons ajoutée) une parcelle de riz lorsqu'elle a au moins une parcelle voisine dans l'un des états que nous venons d'indiquer.

On remarquera que nous sommes là dans une situation de voisinage multiple décrite en 3.3.5 et nous avons fait usage d'un opérateur d'agrégation pour faire en quelque sorte une synthèse des états voisins d'une parcelle donnée. En l'occurence, c'est un opérateur logique (un OU) qui fait office d'opérateur d'agrégation :

```
affect boolean OrOp(group[boolean] gr) {
  boolean result = true;
  for (v in gr) { result = result || v; }
```

FIG. 6.8 – Graphe de voisinage basé sur l'adjacence des parcelles et utilisé par la relation `NextToRice`.

```
  return result;
}

. . .

relation NextToRice [left, right] {
  service testRice() {
    boolean sw = true;
    if ((left.lct == 1) && ((right.lct == 1) || (right.lct == 3)))
        {
      sw = false;
    }
    left.sel <=OrOp= sw;
  }
}
```

Un appel a ce service `testRice()` est effectué à chaque pas de temps sur le graphe de voisinage, juste avant l'appel à `changeLC()` sur le graphe de la relation `isPartOf`.

```
scenario Kottoli {
 . . .
  for (year in y) {
    print(year);
    r.outputKml(year, kml);
    ng.testRice();
    r.changeLC(step);
    step = step+1;
  }
 . . .
}
```

Le service `changeLC()` a dû être adapté pour tenir compte du marquage des parcelles de riz et n'effectuer un changement d'état que lorsque les contraintes sont bien respectées, c'est à dire lorsque la propriété `sel` de la parcelle est *vrai*.

6.4.6 Discussion

L'adaptation en Ocelet de ce modèle nous a permis d'expérimenter un cas où les règles de transitions prennent la forme d'automates à états finis. En tant que langage, Ocelet offre une certaine capacité d'expression dont nous avons profité pour proposer des solutions sur mesure, vis à vis de chacune des hypothèses de ce modèle. Nous avons cependant de fortes chances d'être régulièrement confrontés à des modèles faisant usage d'automates à états finis pour spécifier la dynamique d'un système, et il apparait souhaitable de chercher à généraliser cet aspect du modèle.

Nous pourrions en particulier définir un format de fichier pratique pour décrire des automates, et doter Ocelet d'un datafacer spécialisé dans la lecture de ce type de fichier.

6.5 Conclusion

Les expériences de modélisation que nous avons réalisées avec Ocelet ont permis de tester notre approche de la modélisation, les concepts sur lesquels elle est basée, mais aussi le langage lui même, et l'environnement de modélisation. Ce travail a aussi été l'occasion d'écrire une première série de datafacers donnant ainsi accès à des bases de données géographiques, à des fichiers de type *shapefile*, à des tableaux au format *csv*, à l'exportation de résultats de simulation en *KML*.

Nous n'avons pas rencontré d'obstacle sur le fond pour réaliser ces modèles. Nous avons pu à chaque fois aborder la conception d'un modèle en raisonnant en terme de graphes d'interaction, c'est à dire en nous posant les questions *qui interagit avec qui ?* et *que se passe-t-il lors de ces interactions ?*. Ocelet a permis de réellement prendre en compte les composantes spatiales et leur influence sur le comportement des modèles, sans avoir fixé une forme particulière de représentation de l'espace à priori. Nous avons en outre pu conserver des liens entre les éléments des graphes et l'information géographique nécessaire pour produire des cartes dynamiques en résultat de simulation.

La forme langage métier que nous avons donné à notre outil permet effectivement de s'adapter à une grande diversité de situations de modélisation. Nous avons pu constater en travaillant sur ces modèles, que notre approche de la modélisation incite à apporter un regard particulier sur la façon d'aborder un problème. Cette façon de raisonner n'est pas forcément immédiate au premier abord, mais elle s'aquiert rapidement et l'on aboutit à des solutions originales.

Chapitre 7

Conclusions

Sommaire

7.1 Synoptique et apports de la thèse

Nous avons conçu une nouvelle approche de modélisation dédiée à la simulation de dynamiques spatiales, en visant des applications dans les domaines des sciences de l'environnement. Les concepts à la base de cette approche ont été formalisés, et ils ont constitué les éléments de base d'un langage métier, nommé *Ocelet*, que nous avons spécifié. Les outils permettant la mise en oeuvre de ce langage ont été développés et intégrés sous la forme d'un environnement de modélisation et de simulation. Enfin nous avons pu expérimenter notre nouvelle approche de modélisation et le langage Ocelet à travers la réalisation de plusieurs modèles présentant des situations variées, impliquant des dynamiques paysagères.

7.1.1 Conception et formalisation

Si l'on considère qu'un système est constitué d'entités qui interagissent les unes avec les autres, on peut s'en faire une représentation, c'est à dire un modèle, en choisissant un point de vue qui favorisera l'analyse de certains aspects de ce système.

Pour étudier les relations entre certaines grandeurs caractéristiques révélatrices du comportement général du système, on fera le choix d'une approche descendante de modélisation. On cherchera alors à caractériser globalement les interactions sous la forme d'équations différentielles entre les grandeurs observées.

Quand à l'inverse on peut se faire une idée des règles qui régissent le comportement de chaque entité du système, considérée comme un individu indépendant, on fera le choix d'une approche ascendante de modélisation. En simulant les interactions d'un grand nombre de ces entités, on pourra observer l'émergence de certains comportements de groupe qui auraient été inaccessibles à travers une simple analyse des composants du système.

Les entités d'un système ne sont pas forcément toutes en interaction les unes avec les autres. Les liens qui portent ces interactions constituent des structures qu'il est utile de caractériser parce qu'elles ont une influence non négligeable sur le comportement du système. Nous prétendons que ces structures dépendent de la nature des interactions (hiérarchique, fonctionnelle, spatiale ou sociale par exemple), et qu'il n'y a pas de raison de favoriser à priori un type d'interaction, donc une forme de structure, plutôt qu'une autre.

Nous avons adopté une posture en quelque sorte intermédiaire entre les approches descendante et ascendante de modélisation, en focalisant la modélisation sur les niveaux auxquels s'organisent les interactions. Cela ne signifie pas que nous ayons éliminé le niveau des individus, ni celui d'une vision globale sur le système, mais nous leur avons donné une place qui est relative aux niveaux intermédiaires qui nous semblent davantage structurants. Ce choix permet en outre de placer les structures spatiales au même niveau que les autres, sans choisir à priori une forme de représentation de l'espace qui pourrait s'avérer contraignante.

Nous avons ainsi défini de manière formelle des concepts de modélisation pour chaque niveau :

Individus On y trouve les concepts d'*entité* et de *propriété*, qui peuvent représenter l'état de certains éléments du système.

Interactions Nous avons montré comment combiner la structure qui porte des interactions, et la sémantique associée à la nature de ces interactions, dans un concept nommé *graphe d'interaction* qui est un élément central dans notre approche de la modélisation. La spécification de ces graphes d'interaction se fait à travers la définition de *relations* entre des catégories d'entités. Ces catégories sont basées sur les *rôles* que les entités peuvent jouer lorsqu'elle interagissent les unes avec les autres.

Système global Le système dans son ensemble est modélisé à travers le concept de *scénario*, où l'on spécifie l'état initial du modèle, et l'ordre dans lequel s'enchaînent les opérations qui feront évoluer l'état du modèle au cours d'une simulation. Nous avons également défini le concept de *datafacer* dont le rôle est d'aider à faire le lien entre sources de données et graphes d'interaction d'une part, et entre ces graphes et les différents formats de sortie des simulations (dont la cartographie dynamique)

d'autre part.

7.1.2 Spécification et outillage du langage Ocelet

Nous avons spécifié les éléments du langage métier *Ocelet* dédié à la modélisation et la simulation de dynamiques spatiales. Les principaux concepts que nous avons imaginés pour construire et manipuler des graphes d'interaction sont représentés sous la forme de mots clés du langage (`entity`, `relation`, `scenario`, `datafacer`, ...). La sémantique associée à ces concepts est donc intégrée au langage lui même : par exemple il suffit d'une instruction en Ocelet pour appeler une fonction de transition sur un graphe d'interaction, et cela a pour effet d'exécuter la fonction simultanément sur tous les arcs du graphe. Ce caractère simultané de l'opération est garanti par la sémantique du langage, le modélisateur n'a pas à programmer lui même les parcours de graphes.

Un modèle écrit en Ocelet comporte en général trois parties principales :

1. la définition des types d'entités qui vont être manipulés dans le modèle
2. la définition des relations et des rôles que vont jouer les entités dans un graphe d'interactions
3. la définition d'au moins un scénario dans lequel on construit l'état initial du modèle et où l'on programme l'enchaînement des interactions qui vont faire évoluer son état.

Nous avons conçu Ocelet avec la volonté de le rendre réellement opérationnel. Nous avons donc entamé le travail d'ingénierie nécessaire pour outiller ce langage. Une première version du compilateur et d'un générateur de code ont été réalisés. Le générateur de code produit un programme en langage Java qui fait appel à un ensemble de classes que nous avons développées directement en Java et qui constituent le moteur d'exécution.

Enfin pour faciliter le travail de modélisation avec Ocelet, nous avons construit un environnement de modélisation qui intègre ces outils et qui est basé sur le module *Rich Client Platform* d'Eclipse.

7.1.3 Expérimentations

Nous avons expérimenté notre approche de modélisation par la réalisation de différents modèles, sur des questions de dynamiques paysagères pour lesquelles les structures spatiales ont une influence sur l'évolution du phénomène que l'on souhaite simuler. Ces expériences ont d'abord été réalisées sur le papier, ce qui a permis de guider certains de nos choix dans la conception du langage, puis avec Ocelet et son environnement de modélisation en utilisant les éléments du langage actuellement implémentés. Dans ce document nous avons rendu compte des trois expériences suivantes :

Dissémination de phytopathogènes par voie aérienne entre parcelles cultivées
Nous nous sommes inspirés du mode de dissémination aérien de phytopathogènes pour imaginer un modèle simplifié qui reproduit une dynamique de contamination en se basant sur un graphe de voisinage. Ce graphe sert de structure à un graphe

d'interaction qui porte les règles de diffusion d'un pathogène, en fonction de quelques critères comme la direction et la force du vent et d'un coefficient de voisinage qui sert à pondérer les interactions entre les parcelles. La réalisation de ce modèle a aussi été l'occasion d'ajouter un datafacer donnant accès à des données géographiques stockées dans une base de données (de type PostgreSQL/PostGIS).

Dynamique côtière d'un écosystème de mangrove Il s'agit de simuler la dynamique côtière de la mangrove sur une partie de la côte Guyanaise, située entre Cayenne et Kourou. Ce modèle met en oeuvre un processus d'érosion du trait de côte d'une part, et d'accumulation de vase d'autre part pour tenter d'apréhender la façon dont ce trait de côte a évolué de 1986 à aujourd'hui.

Cet exemple nous a permis d'expérimenter Ocelet dans une situation où l'on cherche à déplacer et déformer des objets ayant une structure spatiale.

Nous avons intégré dans le modèle trois mécanismes d'interaction distincts qui ont été modélisés chacun avec un graphe d'interaction différent. Cette séparation explicite permet de ne réfléchir que sur un processus d'interaction à la fois, et devrait faciliter l'évolution et l'amélioration du modèle.

Evolution de l'usage du sol dans le sud de l'Inde Nous avons reproduit un modèle d'évolution historique de l'usage des sols, dont les règles sont exprimées sous la forme d'automates. Avec cet exercice nous avons voulu éprouver la capacité d'expression d'Ocelet par l'intégration progressive d'hypothèses de modélisation de natures très différentes : on a d'abord un seul automate qui exprime des probabilités de changements d'états, ensuite cet automate peut changer au cours de la simulation, en troisième hypothèse on introduit plusieurs automates qui régissent les changements d'états de différentes portions de la zone d'étude, et enfin on fait dépendre certaines transitions de l'état de parcelles voisines.

L'expérience a montré qu'il n'y avait pas de difficulté particulière à exprimer ces différentes hypothèses avec Ocelet. L'usage de graphes d'interaction peut d'ailleurs apporter des solution originales, à l'exemple de la troisième hypothèse qui a été modélisée simplement en changeant la structure d'un graphe.

7.2 Perspectives

7.2.1 Des primitives plus évoluées

L'approche de modélisation de nous avons développée dans cette thèse se veut générique dans le sens où elle permet de simuler des dynamiques spatiales de toutes natures. Nous avons dû, pour accéder à cette généricité, nous baser sur des concepts très élémentaires. Nous considérons qu'il s'agissait là d'une étape nécessaire, et nous espérons pouvoir nous appuyer sur ces concepts élémentaires pour construire des primitives de modélisation plus évoluées et plus proches des thématiques concernées par la modélisation de dynamiques spatiales.

Lors de la conception d'Ocelet, nous avons gardé à l'esprit cet objectif et cela a orienté certains de nos choix. Le temps nous a manqué pour vérifier si Ocelet permet effectivement

de produire des primitives de modélisation plus évoluées par assemblage de ses éléments de base, et c'est un travail dans lequel nous souhaitons nous engager. Il nous semble qu'il y a d'ailleurs de véritables questions de recherches associées à la construction de telles primitives, en particulier dans les liens que l'on pourrait établir entre des structures de graphe particulières et des situations récurrentes de modélisation.

7.2.2 Représentation des connaissances

La description de primitives de modélisation sous la forme d'ontologies constitue une autre voie possible pour se rapprocher des besoins de certaines thématiques, mais aussi pour améliorer nos outils. En faisant par exemple usage d'un langage de définition d'ontologies (comme OWL), on se doterait d'une capacité à raisonner sur les constituants d'un modèle.

Cela peut s'avérer intéressant dans le contexte de la modélisation environnementale où ce sont souvent plusieurs domaines de compétences qu'il faut réunir. Chaque domaine apporte son point de vue, son expertise, et ses éléments de modèles (sous la forme de primitives évoquées au paragraphe précedent). L'assemblage de sous-modèles qui pris séparement sont consistants peut aboutir à un modèle d'ensemble qui ne l'est pas. Un raisonneur peut aider à détecter des inconsistances.

L'usage d'ontologies pourrait d'ailleurs nous aider à évoluer vers un environnement davantage graphique d'aide à la modélisation. Il serait en effet important de bénéficier de l'intégration d'un raisonneur dans une telle interface.

Ces premières réflexions sur l'usage d'ontologies dans Ocelet ont fait l'objet d'une communication d'O. Curé et al. [41] en 2010.

7.2.3 Evolution du langage

Les comportements portés par les graphes d'interaction sont définis de manière déclarative dans Ocelet. Il s'agit bien entendu d'un choix délibéré de notre part, d'abord pour libérer les modélisateurs des détails d'implémentation relatifs aux parcours de graphes ou à l'aspect simultané des fonctions de transition, mais aussi pour avoir la possibilité d'apporter certaines optimisations a postériori, prenant en compte le contexte d'exécution d'un modèle.

Nous avons l'intention d'étudier plusieurs formes d'optimisations concernant les graphes d'interaction. Nous pensons par exemple que dans les situations où l'on fait usage d'un graphe de voisinage structuré en motifs réguliers (comme une grille, ou une triangulation) on pourrait optimiser l'exécution de fonctions de transition en tirant parti des caractérisitiques de ces structures.

La gestion du temps dans un modèle écrit en Ocelet est spécifiée dans un scénario, sous la forme d'un programme séquentiel. Nous pourrions explorer des modes de description de l'enchainement des opérations sous formes de règles qui permettraient d'aller vers un langage entièrement déclaratif. Cela pourrait offrir aux modélisateurs diverses options

dans la gestion du temps, sans que ceux-ci aient à se soucier de la façon de les programmer.

7.2.4 L'expérimentation source d'innovation

Enfin nous ne pouvons terminer cette section sans revenir à l'aspect finalisé du travail de recherche dans lequel nous sommes engagé. Nous avons en effet l'intuition que les perspectives les plus importantes vont prendre leur source dans l'expérimentation, sur une large diversité d'applications, avec des problématiques souvent pluridisciplinaires. Le laboratoire dans lequel nous exerçons est d'ailleurs un cadre privilégié pour confronter nos concepts de modélisation à de nombreuses problématiques de terrain, dans lesquelles la composante spatiale est prise en compte dans ses diverses formes.

Chaque exercice de modélisation d'un aspect du réel est particulier et apporte son lot de questions singulières, parfois inattendues. L'approche de la modélisation que nous avons développé, qui donne une place centrale aux niveau des intéractions et aux structures qui les portent, contient une originalité qui permettra nous l'espérons, d'aborder certaines questions avec un oeil nouveau. La forme d'un langage métier que nous avons choisi offre en outre une importante capacité d'adaptation. Cela dit, nous ne doutons pas que certaines situations de modélisation que l'expérimentation nous apportera, remettront en question certains aspects de nos travaux, et nous forcera à imaginer de nouvelles façons d'y répondre. Pour dire la vérité, nous l'espérons. La part de créativité dont il est possible de faire preuve dans l'exercice de la recherche est en effet le principal moteur qui anime notre travail.

Bibliographie

[1] The eclipse foundation. http ://www.eclipse.org/.

[2] Eclipse rich client platform. http ://www.eclipse.org/home/categories/rcp.php/.

[3] Open services gateway initiative. http ://www.osgi.org.

[4] Tatoo compiler compiler. http ://tatoo.univ-mlv.fr/.

[5] Postgresql, postgis spatial extension. http ://postgis.refractions.net/, 2001.

[6] Meta object facility (mof) 2.0 core specification. http ://www.omg.org/mda/specs.htm#mof, 2004.

[7] Omg unified modeling language, 2010.

[8] Open geospatial consortium. http ://www.opengeospatial.org, 2011.

[9] Ayoub Ait Lahcen, Pascal Degenne, Danny Lo Seen, and Didier Parigot. Developing a service-oriented component framework for a landscape modeling language. In *Software Engineering and Applications*, number 669, Cambridge, Massachussets, USA, 2009. Acta Press.

[10] Julie Alet. What are the possible current and future dynamics of agroforestry and agricultural landscapes in the Western Ghats (southern India) ? A case study : Kottoli, a village in Coorg. Master's thesis, AgroParisTech - Engref, French Institute of Pondicherry, India, 2011.

[11] Matthieu Amiguet. *Un modèle componentiel dynamique pour les systèmes multi-agents organisationnels*. PhD thesis, Université de Neuchâtel, 2003.

[12] Edward J. Anthony, Antoine Gardel, Nicolas Gratiot, Christophe Proisy, Mead A. Allison, Franck Dolique, and François Fromard. The amazon-influenced muddy coast of south america : A review of mud-bank–shoreline interactions. *Earth-Science Reviews*, 103 :99–121, 2010.

[13] W. Ross Ashby. *An introduction to cybernetics*. Chapman&Hall Ltd., London, 1956.

[14] F. Aurenhammer and R. Klein. *Handbook of Computational Geometry*, chapter Voronoi diagrams, pages 201–290. Elsevier Science Publishing, 2000.

[15] F. Baltzer, M. Allison, and F. Fromard. Material exchange between the continental shelf and mangrove-fringed coasts with special reference to the amazon ?guianas coast. *Marine Geology*, 208(2-4) :115–126, 2004.

[16] Jorgen Bang-Jensen and Gregory Gutin. *Digraphs : Theory, Algorithms and Applications*. Springer, August 2002.

[17] Fernando J. Barros. Dynamic structure discrete event system specification : a new formalism for dynamic structure modeling and simulation. In *WSC '95 : Proceedings of the 27th conference on Winter simulation*, pages 781–785, Washington, DC, USA, 1995. IEEE Computer Society.

[18] Claude Berge. *Graphes et Hypergraphes*. Dunod, 1970.

[19] Uta Berger, Hanno Hildenbrandt, and Volker Grimm. Towards a standard for the individual-based modeling of plant populations : self-thinning and the field of neighborhood approach. *Natural Resource Modeling*, 15(1) :39–54, 2002.

[20] J.E. Bergez, P. Chabrier, F. Garcia, C. Gary, D. Makowski, G. Quesnel, E. Ramat, H. Raynal, N. Rousse, and D. Wallach. Record : a new software platform to model and simulate cropping systems. In *Farming System Design*, Monterey, CA, August 2009. IEMSS.

[21] Alan A. Berryman. The origins and evolution of predator-prey theory. *Ecology*, 73(5) :pp. 1530–1535, 1992.

[22] Ludwig von Bertalanffy. An outline of general system theory. *The British Journal for the Philosophy of Science*, 1(2) :134–165, August 1950.

[23] Georges Bertrand. Le paysage et la géographie : un nouveau rendez-vous. *Treballs de la Societat Catalana de Geografia*, (50), 2002.

[24] J A Bondy and U S R Murty. *Graph theory with applications*, volume 290. MacMillan London, 1976.

[25] Andrei Borshchev and Alexei Filippov. From system dynamics and discrete event to practical agent based modeling : Reasons, techniques, tools. In *The 22nd International Conference of the System Dynamics Society*, 2004.

[26] François Bousquet, Innocent Bakam, Hubert Proton, and Christophe Le Page. Cormas : Common-pool resources and multi-agent systems. In Angel Pasqual del Pobil, José Mira, and Moonis Ali, editors, *Tasks and Methods in Applied Artificial Intelligence*, volume 1416 of *Lecture Notes in Computer Science*, pages 826–837. Springer Berlin / Heidelberg, 1998.

[27] François. Bousquet and Christophe Le Page. Multi-agent simulations and ecosystem management : a review. *Ecological Modelling*, 176(3-4) :313 – 332, 2004.

[28] Roger Brunet. La composition des modèles dans l'analyse spatiale. *L'Espace Géographique*, (4) :253–264, 1980.

[29] F. Burel and J. Baudry. *Ecologie du paysage : concepts, méthodes et applications.* Lavoisier, Paris, 1999.

[30] Yvan Bédard, Suzie Larrivée, Marie-Josée Proulx, and Martin Nadeau. *Modeling Geospatial Databases with Plug-Ins for Visual Languages : A Pragmatic Approach and the Impacts of 16 Years of Research and Experimentations on Perceptory*, volume 3289, pages 17–30. Springer, 2004.

[31] Margot D. Cantwell and Richard T. T. Forman. Landscape graphs : Ecological modeling with graph theory to detect configurations common to diverse landscapes. *Landscape Ecology*, 8 :239–255, 1993. 10.1007/BF00125131.

[32] Julien Cervelle, Rémi Forax, and Gilles Roussel. Tatoo : An innovative parser generator. In Ralf Gitzel, Markus Alesky, Matin Schader, and Chandra Krintz, editors, *4th International Conference on Principles and Practices of Programming in Java (PPPJ'06)*, ACM International Conference Proceedings, pages 13–20, 2006.

[33] Alex Chung Hen Chow and Bernard P. Zeigler. Revised devs : a parallel, hierarchical, modular, modeling formalism. In *WSC '94 : Proceedings of the 26th conference on Winter simulation*, pages 716–722, San Diego, CA, USA, 1994. Society for Computer Simulation International.

[34] Christophe Claramunt and Marius Thieriault. *Managing Time in GIS An Event-Oriented Approach*, pages 23–42. Springer-Verlag, 1995.

[35] Vincent Clément. Contribution épistémologique à l'étude du paysage. *Mélanges de la Casa de Velàsquez*, 30-3 :221–237, 1994.

[36] Philippe Clergeau and Guy Désiré. Biodiversité, paysage et aménagement du corridor à la zone de connexion biologique. *Mappemonde*, 55(3) :19–23, 1999.

[37] Robert Costanza. Simulation modeling on the macintosh using stella. *BioScience*, 37 :129–132, 1987.

[38] Robert Costanza and Alexey Voinov. *Landscape Simulation Modeling. A spatially explicit, dynamic approach.* Springer-Verlag New York, 2004.

[39] Carine Courbis, Pascal Degenne, Alexandre Fau, and Didier Parigot. Un modèle abstrait de composants adaptables. *revue TSI, Composants et adaptabilité*, 23(2) :231–252, 2004.

[40] Luciano R. Coutinho, Jaime S. Sichman, and Olivier Boissier. Modeling organization in mas : A comparison of models. In *First Workshop on Software Engineering for Agent-oriented Systems*, 2005.

[41] Olvier Curé, Rémi Forax, Pascal Degenne, Didier Parigot, and Ait Lahcen Ayoub. Ocelet : An ontology-based domain specific language to model complex domains. In *International Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS) June 13-19, 2010 - Athens/Glyfada, Greece*, 2010.

[42] William S Currie. Units of nature or processes across scales ? the ecosystem concept at age 75. *New Phytologist*, 190 :21–34, 2011.

[43] Michael A.B. Deakin. A standard form for the kermack-mckendrick epidemic equations. *Bulletin of Mathematical Biology*, 37 :91 – 95, 1975.

[44] Donald L DeAngelis and Wolf M Mooij. Individual-based modeling of ecological and evolutionary processes1. *Annual Review of Ecology Evolution and Systematics*, 36(1) :147–168, 2005.

[45] P. Degenne, D. Lo Seen, D. Parigot, R. Forax, A. Tran, A. Ait Lahcen, O. Curé, and R. Jeansoulin. Design of a domain specific language for modelling processes in landscapes. *Ecological Modelling*, July 2009.

[46] Pascal Degenne, Didier Parigot, Olvier Curé, Ayoub Ait Lahcen, Rémi Forax, and Danny Lo Seen. Modelling the environment using graphs with behaviour : do you speak ocelet ? In *International Congress on Environmental Modelling and Software*, Ottawa, Canada, 2010.

[47] Richard Diestel. *Graph Theory*. Springer-Verlag, New York, 2005.

[48] Raphael Duboz. *Intégration de modèles hétérogènes pour la modélisation et la simulation de systèmes complexes, Application à la modélisation multi-échelles en écologie marine*. PhD thesis, Université du Littoral - Côte d'Opale, 2004.

[49] Joshua M. Epstein and Robert L. Axtell. *Growing Artificial Societies : Social Science from the Bottom Up*. The MIT Press, 1st printing edition, 1996.

[50] Andrew Fall and Joseph Fall. A domain-specific language for models of landscape dynamics. *Ecological Modelling*, 141(1-3) :1 – 18, 2001.

[51] Andrew Fall, Marie-Josee Fortin, Micheline Manseau, and Dan OBrien. Spatial graphs : Principles and applications for habitat connectivity. *Ecosystems*, 10 :448–461(14), April 2007.

[52] Jacques Ferber. *Les systèmes multi-agents, vers une intelligence collective*. InterEditions, 1995.

[53] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations : An organizational view of multi-agent systems. In *In : LNCS n. 2935 : Procs. of AOSE'03*, pages 214–230, 2003.

[54] Jean Charles Filleron. “le paysage, cela existe, même lorsque je ne le regarde pas” ou quelques réflexions sur les pratiques paysagères des géographes. In *Journées scientifiques ”Le paysage entre culture et nature”, Pôle universitaire européen de Montpellier*, 1998.

[55] Jerôme Fiot and Nicolas Gratiot. Structural effects of tidal exposures on mudflats along the french guiana coast. *Marine Geology*, 228(1-4) :25–37, 2006.

[56] Richard T. T. Forman and Michel Godron. Patches and structural components for a landscape ecology. *BioScience*, 31(10) :733–740, 1981.

[57] Jay W. Forrester. *Principles of Systems*. Wright-Allen Press Cambridge, Mass., 1968.

[58] Jay W. Forrester. *Urban dynamics*. M.I.T. Press Cambridge, Mass., 1969.

[59] Jay W. Forrester. *World dynamics*. Wright-Allen Press Cambridge, Mass., 1971.

[60] Stan Franklin and Art Graesser. Is it an agent, or just a program ? : A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[61] J.M. Froidefond, M. Pujos, and X. Andre. Migration of mud banks and changing coastline in french guiana. *Marine Geology*, 84(1-2) :19 – 30, 1988.

[62] F Fromard, C Vega, and C Proisy. Half a century of dynamic coastal change affecting mangrove shorelines of french guiana. a case study based on remote sensing data analyses and field surveys. *Marine Geology*, 208(2-4) :265–280, 2004.

[63] Galilei Galileo. *Dialogue sur les deux grands systèmes du monde*. 1632.

[64] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[65] Antoine Gardel and Nicolas Gratiot. A satellite image-based method for estimating rates of mud bank migration, french guiana, south america. *Journal of Coastal Research*, 214(1978) :720–728, 2005.

[66] Martin Gardner. The fantastic combinations of john conway’s new solitaire game ”life”. *Scientific American*, 223 :120–123, 1970.

[67] Robert Gardner, Bruce Milne, Monica Turnei, and Robert O’Neill. Neutral models for the analysis of broad-scale landscape pattern. *Landscape Ecology*, 1 :19–28, 1987.

[68] Les Gasser. Organizations in multi-agent systems. In *Pre-Proceeding of the 10th European Worshop on Modeling Autonomous Agents in a Multi-Agent World (MAA-MAW’2001)*, Annecy, 2001.

[69] Les Gasser, Carl Braganza, and Nava Herman. *Distributed Artificial Intelligence*, chapter MACE A Flexible Testbed for Distributed AI Research. Pitman, 1987.

[70] Cédric Gaucherel, Nathalie Giboire, Valérie Viaud, Thomas Houet, Jacques Baudry, and Françoise Burel. A domain-specific language for patchy landscape modelling : The brittany agricultural mosaic as a case study. *Ecological Modelling*, 194(1-3) :233 – 243, 2006.

[71] Cédric Gaucherel, Sébastien Griffon, Laurent Misson, and Thomas Houet. Combining process-based models for future biomass assessment at landscape scale. *Landscape Ecology*, 25 :201–215, 2010. 10.1007/s10980-009-9400-6.

[72] Jean-Louis Giavitto and Olivier Michel. Mgs : a rule-based programming language for complex objects and collections. *Electronic Notes in Theoritecal Computer Science*, 59(4) :286 – 304, 2001.

[73] Jean-louis Giavitto and Olivier Michel. Data structure as topological spaces. In *In Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02*, pages 137–150, 2002.

[74] Jean-Louis Giavitto and Antoine Spicher. *La Morphogénèse*, chapter Morphogénèse informatique, pages 328–352. Belin, Paris, 2006.

[75] Micheal F Goodchild. A spatial analytical perspective on geographical information systems. *International Journal of Geographical Information Systems*, 1(4) :327–334, 1987.

[76] Nicolas Gratiot, Antoine Gardel, and Edward J Anthony. Trade-wind waves and mud dynamics on the french guiana coast, south america : Input from era-40 wave data and field investigations. *Marine Geology*, 236 :15 – 26, 2007.

[77] V. Grimm. Ten years of individual-based modelling in ecology : what have we learned and what could we learn in the future ? *Ecological Modelling*, 115(2-3) :129–148, February 1999.

[78] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jørgensen, Wolf M. Mooij, Birgit Muller, Guy Pe'er, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja Rüger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vabø, Ute Visser, and Donald L. Deangelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2) :115–126, September 2006.

[79] Volker Grimm, Uta Berger, Donald L. DeAngelis, J. Gary Polhill, Jarl Giske, and Steven F. Railsback. The odd protocol : A review and first update. *Ecological Modelling*, 221(23) :2760 – 2768, 2010.

[80] E J Gustafson. Quantifying landscape spatial pattern : What is the state of the art ? *Ecosystems*, 1(2) :143–156, 1998.

[81] Olivier Gutknecht. *Proposition d'un modèle organisationnel générique de systèmes multi-agents*. PhD thesis, Université de Montpellier II, 2001.

[82] Olivier Gutknecht and Jacques Ferber. Madkit : a generic multi-agent platform. In *Proceedings of the fourth international conference on Autonomous agents*, AGENTS '00, pages 78–79, New York, NY, USA, 2000. ACM.

[83] Carl Hewitt. Viewing control structures as patterns of passing messages. Technical report, Massachussets Institute of Technology, December 1976.

[84] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence : Advances in Artificial Intelligence*, SBIA '02, pages 118–128, London, UK, UK, 2002. Springer-Verlag.

[85] IEEE, http ://www.fipa.org/index.html. *FIPA : Foundation for Intelligent Physical Agents*, 2005.

[86] Marton Ivanyi, Rajmund Bocsi, Laszlo Gulyas, Vilmos Kozma, and Richard Legendi. The multi-agent simulation suite. *Emergent Agents and Socialities : Social and Organizational Aspects of Intelligence (AAAI Fall Symposium Series 2007)*, 2007.

[87] Nicholas R. Jennings. On agent-based software engineering. *Artif. Intell.*, 117 :277–296, March 2000.

[88] T H Keitt, D L Urban, and B T Milne. Detecting critical scales in fragmented landscapes. *Conservation Ecology*, 1(1) :4, 1997.

[89] W. O. Kermack and A. G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A*, 115(772) :700–721, 1927.

[90] G. C. Kineke, R. W. Sternberg, J. H. Trowbridge, and W. R. Geyer. Fluid-mud processes on the amazon continental shelf. *Continental Shelf Research*, 16(5-6) :667–696, 1996.

[91] Benjamin Kuipers. *Qualitative Reasoning*. The MIT Press, 1994.

[92] Gail Langran. *Time in Geographic Information Systems.* Taylor & Francis, London, 1992.

[93] Jean-Louis Le Moigne. *La Théorie du Système Général, Théorie de la Modélisation*. PUF, nouvelle édition complétée en 2006 disponible sur internet à http ://www.mcxapc.org/ouvrages.php ?a=display&id=48 edition, 1977.

[94] Christophe Le Page, François Bousquet, Innocent Bakam, Alassane Bah, and Christian Baron. Cormas : A multiagent simulation toolkit to model natural and social dynamics at multiple scales. In *The ecology of scales*, Wageningen, The Netherlands, June 27-30 2000.

[95] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason : A multiagent simulation environment. *SIMULATION*, 81(7) :517–527, 2005.

[96] Laurent Magnin. *Modélisation et simulation de l'environnement dans les systèmes multi-agents*. PhD thesis, Université Pierre et Marie Curie (Paris 6), Novembre 1996.

[97] Thomas R. Malthus. *An Essay on the Principle of Population*. 1798.

[98] Margaret L Margosian, Karen A Garrett, J M Shawn Hutchinson, and Kimberly A With. Connectivity of the american agricultural landscape : Assessing the national risk of crop pest and disease spread. *BioScience*, 59(2) :141–151, 2009.

[99] Tom Maxwell and Robert Costanza. A language for modular spatio-temporal simulation. *Ecological Modelling*, 103(2-3) :105 – 113, 1997. To Read.

[100] K. McGarigal, S. A. Cushman, M. C. Neel, and E. Ene. *FRAGSTATS : Spatial Pattern Analysis Program for Categorical Maps.* University of Massachusetts, Amherst. http ://www.umass.edu/landeco/research/fragstats/fragstats.html, 2002.

[101] Jeremy Mennis, Donna Peuquet, and Liujian Qian. A conceptual framework for incorporating cognitive principles into geographical database representation. *International Journal of Geographical Information Science*, 14(6) :501–520, 2000.

[102] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4) :316–344, 2005.

[103] Nelson Minar, Roger Burkhart, Chris Langton, and Manor Askenazi. The swarm simulation system : A toolkit for building multi-agent simulations. *Simulation*, (96-06-042) :1–11, 1996.

[104] Emily S Minor and Dean L Urban. Graph theory as a proxy for spatially explicit population models in conservation planning. *Ecological Applications*, 17(6) :1771–1782, 2007.

[105] Marvin Minsky. *La société de l'esprit*. InterEditions, Paris, 1988.

[106] André Miralles. *Ingénierie des modèles pour les applications environnementales*. PhD thesis, Université de Montpellier, 2006.

[107] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. The Johns Hopkins University Press, 2001.

[108] Edgar Morin. *Introduction à la pensée complexe*. Seuil, 2005.

[109] R.I Muetzelfeldt, , and J. Massheder. The simile visual modelling environment. *European Journal of Agronomy*, 18 :345–358, 2003.

[110] Jean-Pierre Müller. The mimosa generic modelling and simulation platform : The case of multi-agent systems. In *SCS. Proceedings of the 5th Workshop on Agent-Based Simulation*, pages 77–86, Lisbon, Portugal., 2004.

[111] Jean-Pierre Müller. Towards a formal semantics of event-based multi-agent simulations. In Nuno David and Jaime Simão Sichman, editors, *Multi-Agent-Based Simulation IX*, pages 110–126. Springer-Verlag, Berlin, Heidelberg, 2009.

[112] Pierre-Alain Muller, Frédéric Fondement, Benoit Baudry, and Benoit Combemale. Modeling modeling modeling. *SOSYM*, 2010.

[113] J D Murray. *Mathematical Biology I : An Introduction*, chapter Predator–Prey Models : Lotka–Volterra Systems, pages 79–83. Springer, 2002.

[114] Jean-Pierre Müller. Mimosa : using ontologies for modeling and simulation. In *Proceedings of the 8th Asia-Pacific complex systems conference*, 2007.

[115] Jean-Pierre Müller. Mimosa modelling and simulation platform - http ://mimosa.sourceforge.net/index.html, 2010.

[116] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[117] M.J. North, T.R. Howe, N.T. Collier, and R.J. Vos. The repast simphony development environment. In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, Argonne National Laboratory, Argonne, IL USA, 2005.

[118] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations : Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[119] David O'Sullivan. *Graph-based cellular automaton models of urban spatial processes*. PhD thesis, University College London, 2008.

[120] C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications : The Mads Approach*. Springer, 2006.

[121] Miles T. Parker. What is ascape and why should you care ? *Journal of Artificial Societies and Social Simulation*, 4(1), 2001.

[122] Donna J. Peuquet. It's about time : A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84(3) :441–461, 1994.

[123] Donna J. Peuquet. *Representations of Space and Time*. The Guilford Press, London, 2002.

[124] C. Proisy, E. Blanchard, A. Ait Lahcen, P. Degenne, and D. Lo Seen. Toward the simulation of the amazoninfluenced mangrove-fringed coasts dynamics using ocelet. In *International Conference on Integrative Landscape Modelling*, Montpellier, France, February 3-5 2010.

[125] Christophe Proisy, Nicolas Gratiot, Edward J Anthony, Antoine Gardel, François Fromard, and Patrick Heuret. Mud bank colonization by opportunistic mangroves : A case study from french guiana using lidar data. *Continental Shelf Research*, 29(3) :632–641, 2009.

[126] Gauthier Quesnel. *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes*. PhD thesis, Université du Littoral - Côte d'Opale, 2006.

[127] Gauthier Quesnel, Raphaël Duboz, and Éric Ramat. The virtual laboratory environment – an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17 :641–653, April 2009.

[128] Bronwyn Rayfield, Marie-Josée Fortin, and Andrew Fall. The sensitivity of least-cost habitat graphs to relative cost surface values. *Landscape Ecology*, 25(4) :519–532, 2009.

[129] F. Reitsma and J. Albrecht. Implementing a new data model for simulating processes. *International Journal of Geographical Information Science*, 19(10) :1073–1090, 2005.

[130] P Rigaux, M Scholl, and A Voisard. *Spatial Databases : With Application to GIS*. Morgan Kaufmann, 2001.

[131] Ronald Ross. *The Prevention of Malaria*. John Murray, London, 2nd edition edition, 1911.

[132] Thomas C. Schelling. Models of segregation. In *The American Economic Review*, volume 59, pages 488 – 493. American Economic Association, May 1969.

[133] Thomas C. Schelling. Dynamic models of segregation. *The Journal of Mathematical Sociology*, 1(2) :143–186, 1971.

[134] R S Schick and S T Lindley. Directed connectivity among fish populations in a riverine network. *Journal of Applied Ecology*, 44(6) :1116–1126, 2007.

[135] Antoine Spicher. *Transformation de collections topologiques de dimension arbitraire. Application à la modélisation de systèemes dynamiques*. PhD thesis, Université d'Évry-Val-d'Essonne, 2006.

[136] Antoine Spicher and Olivier Michel. Using rewriting techniques in the simulation of dynamical systems : Application to the modeling of sperm crawling. In *Fifth International Conference on Computational Science (ICCS'05)*, volume I, pages 820–827, 2005.

[137] Eric A Treml, Patrick N Halpin, Dean L Urban, and Lincoln F Pratson. Modeling population connectivity by ocean currents, a graph-theoretic approach for marine conservation. *Landscape Ecology*, 23(S1) :19–36, 2007.

[138] Monica G Turner. Landscape ecology : What is the state of the science ? *Annual Review of Ecology Evolution and Systematics*, 36(1) :319–344, 2005.

[139] Monica G. Turner, Robert H. Gardner, and Robert V. O'Neill. *Landscape Ecology in Theory and Practice*. Springer, 2001.

[140] Dean Urban and Timothy Keitt. Landscape connectivity : A graph-theoretic perspective. *Ecology*, 82(5) :1205 – 1218, 2001.

[141] Dean L Urban, Emily S Minor, Eric A Treml, and Robert S Schick. Graph models of habitat mosaics. *Ecology Letters*, 12(3) :260–273, 2009.

[142] P.-F. Verhulst. *Correspondance Mathematique et Physique de l'observatoire de Bruxelles*, volume X, chapter Notice sur la loi que la population suit dans son accroissement, pages 113–121. Quetelet, A, 1838.

[143] Christine Voiron. L'espace dans la modélisation des intéractions nature-société. *Actes du colloque Interactions Nature-société, analyse et modèles. La Baule*, 2006.

[144] Christine Voiron and Jean-Pierre Chery. Espace géographique, spatialisation et modélisation en dynamique des systèmes. In *Res-Systemica*, volume 5. UES, 2005. Actes du VIe Congrès Européen de Systémique.

[145] Ludwig von Bertalanffy. *General System theory : Foundations, Development, Applications*. George Braziller, New York, 1968.

[146] M. Wachowicz. *Object-oriented design for temporal GIS*. Research monographs in geographic information systems. Taylor & Francis, 1999.

[147] Gabriel Wainer. *Discrete-Events Cellular Models with Explicit Delays*. PhD thesis, Université d'Aix-Marseille III, 1998.

[148] Gabriel Wainer. Improved cellular models with parallel cell-devs. *TRANSACTIONS of The Society for Modeling and Simulation International*, 17(2) :73–88, June 2000.

[149] Norbert Wiener. *Cybernetics : Or the Control and Communication in the Animal and the Machine*. Hermann, Paris, 1948.

[150] Uri Wilensky. *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (http ://ccl.northwestern.edu/netlogo/), 1999.

[151] K A With and A W King. The use and misuse of neutral landscape models in ecology. *Oikos*, 79(2) :219–229, 1997.

[152] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3) :285–312, 2000.

[153] M F Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37(1) :26–34, 1994.

[154] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation*. Academic press, 2 edition, 2000.

[155] Bernard P. Ziegler. *Theory Of Modeling and Simulation*. Wiley Interscience, 1976.

Index

Table des figures

Notations

Description du langage Ocelet

Dans la description du langage Ocelet, en particulier dans le chapitre 4, nous présentons certains éléments de syntaxe du langage selon la notation suivante :

`'Mot'` Indique que le mot "`Mot`" doit être présent une fois.

`'x'` Indique que le catactère "`x`" doit être présent une fois.

`<expression>` Indique qu'une expression (et une seule) doit impérativement être indiquée à la place de `<expression>`

`[expression]` Indique qu'une expression (et une seule) peut être indiquée à la place de `[expression]` mais que sa présence est facultative.

`expression+` L'expression doit être présente 1 fois ou plus.

`expression*` L'expression peut apparaitre 0 fois ou plus

`expr1 | expr2` L'une ou l'autre des expressions `expr1` ou `expr2` est présente.

Annexes

1 Plateforme de modélisation d'Ocelet

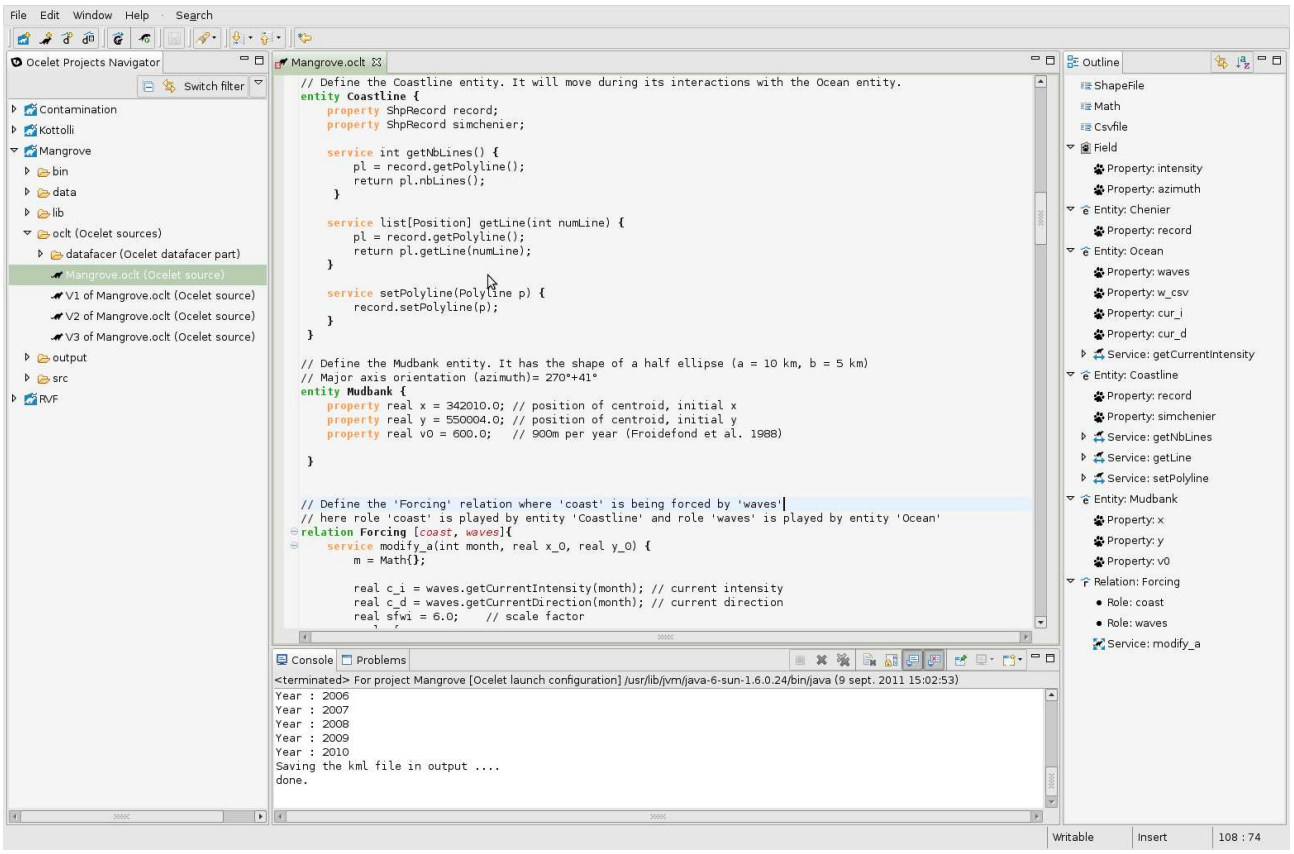


FIG. 1 – Plateforme de modélisation d'Ocelet

2 Grammaire d'Ocelet

Cette grammaire expérimentale est au format *ebnf*, il s'agit du fichier que nous utilisons pour générer l'analyseur lexical, l'analyseur syntaxique et les classes de l'arbre de syntaxe abstraite d'Ocelet à l'aide du générateur Tatoo (voir 5.2).

Une version est maintenue à jour sur le serveur Subversion d'Ocelet à l'adresse suivante :
`http ://svn.ocelet.fr/compiler/ocelet.ebnf`

```
directives:
  autoalias

priorities:
  assign=           1 nonassoc
  range=            2 left
  unary =           3 right
  boolean_and_or =  4 left
  eq =              5 left
  if_else =         6 left
  plus_minus=       7 left
  star_slash=       8 left
  pow =             9 left

tokens:
  plus= '\+'                [plus_minus]
  minus= '-'                [plus_minus]
  star= '\*'                [star_slash]
  slash='\/'                [star_slash]
  mod='%'                   [star_slash]
  pow='\^\^'                [pow]
  band='&&'                 [boolean_and_or]
  bor='\|\|'                [boolean_and_or]
  eq='=='                   [eq]
  neq='!='                  [eq]
  lt='<'                    [eq]
  gt='>'                    [eq]
  le='<='                   [eq]
  ge='>='                   [eq]
  assign= '='               [assign]
  lpar= '\('
  rpar= '\)'
  lcurl='\{'
  rcurl='\}'
  lopt='\['
```

```
ropt='\]'
bang='!'                    [unary]
dot='\.'
range='\.\.'                [range]
colon=':'
semicolon=';'
comma=','


in="in"
_if="if"
_else="else"
_for="for"
_while="while"
_do="do"
_return="return"
_break="break"
_continue="continue"
_this="this"
print='print'
print2file='print2file'
_enum="enum"
group="group"
list="list"
filter="filter"

affect="affect"
structure="structure"
datafacer="datafacer"
entity="entity"
property="property"
service="service"
relation="relation"
scenario="scenario"
_const="const"
global="global"
_uses="uses"

_boolean="boolean"
_int="int"
real="real"
_void="void"
none="\?"
text="text"
```

```
  null_literal='null'
  boolean_literal="true|false"
  value_literal= "([0-9]+(\.([0-9])+)?)|(0[xX][0-9a-fA-F]+)"
  id= "([a-z]|[A-Z]|_)([a-z]|[A-Z]|[0-9]|_)*"
  text_value ="\u022([^\u022]|\\\u022)*\u022"
  hat = "@" [pow]



blanks:
  space = "( |\t|\r|\n)+"

comments:
  cplusplus_comment = "\/\/([^\r\n])*(\r)?\n"
  multiline_comment = "\/\*([^*]|[(\r)?\n]|(\*+([^*/]|((\r)?\n))))*\*+\/"

error:
 error

types:
  'boolean_literal': boolean
  'value_literal': Object
  'id': String
  'text_value': String


starts:
  start

productions:

  primitive_type = 'boolean'    { primitive_type_boolean }
                 | 'int'        { primitive_type_int }
                 | 'real'{ primitive_type_real }
                 ;

  relation_type = 'id' '[' type/'comma'+ ']'   { relation_type }
                ;

  group_type  = 'group' '[' type ']'           { group_type }
              ;

  list_type = 'list' '[' type ']'{ list_type }
  ;
```

```
  text_type = 'text'{ text_type }
;
  type = 'id'                              { type_id }
       | primitive_type                    { type_primitive }
       | relation_type                     { type_relation }
       | group_type                        { type_group }
       | list_type { type_list }
       | text_type { type_text }
       ;


  affect_operator = field_access '<=' 'id' '=' field_access  { affect_operator_def
     ;

  start = script_member*    { start }
        ;

  script_member = const_def        { script_member_const_def }
   | affect_def    { script_affect_def }
                | structure_def    { script_structure_def }
                | datafacer_def    { script_datafacer_def }
                | entity_def       { script_member_entity_def }
                | relation_def     { script_member_relation_def }
                | scenario_def     { script_member_scenario_def }
| uses_def        { script_member_uses_def }
                ;

  affect_def = 'affect' type 'id' '(' parameters ')' block   { affect_def }
   ;


  uses_def = 'uses' 'id' / 'dot' + ';'   { uses_def }
                ;

  datafacer_def =  'datafacer' 'id' / 'dot' + ';'{ datafacer_import_def }
   |  'datafacer' 'id' '{' datafacer_member* '}'{ datafacer_member_def}
        ;


  datafacer_member = service_def { datafacer_member_service_def }
    | 'error'   { datafacer_member_error }
   ;

  structure_def = 'structure' 'id' '{' structure_member* '}'   { structure_def }
       ;
```

```
structure_member = property_def { structure_member_property_def }
 | 'error'       { structure_member_error }
 ;

entity_def = 'entity' 'id' '{' entity_member* '}'  { entity_def }
           ;

entity_member = property_def  { entity_member_property_def }
              | service_def   { entity_member_service_def }
              | 'error'       { entity_member_error }
              ;

property_def = 'property' type 'id' property_init? ';'   { property_def_type }
             | 'property' 'enum' 'id' '=' 'enum' '{' 'id'/'comma'+ '}' ';'
                                                    { property_def_enum }
             | 'property' type 'id' 'hat' '[' 'value_literal' ']' property_init? ';'
                                                  { property_def_history }
             ;

property_init = '=' expr    { property_init }
                ;


relation_def = 'relation' 'id' '[' 'id'/'comma'+ ']' '{' relation_member* '}'
                                                          { relation_def }
             ;

relation_member = service_def    { relation_member_service_def }

                | 'error'        { relation_member_error }
                ;

service_def = 'service' 'id' '(' parameters ')' abstract_or_block
                                                      { service_def_void }
            | 'service'  type 'id' '(' parameters ')' abstract_or_bloc
                                                  { service_def_returntype }
            ;

abstract_or_block = block   { abstract_or_block_block }
                  | ';'     { abstract_or_block_semicolon }
                  ;

parameters = parameter_def/'comma'*   { parameters }
           ;
```

```
parameter_def = type 'id'   { parameter_def }
              ;

scenario_def = 'scenario' 'id' block   { scenario_def }
             ;

const_def = 'const' 'id' '=' expr  ';' { const_def }
          ;

block = '{' instr* '}'      { block }
      ;

instr = declaration ';'           { instr_declaration }
      | filter_def ';'{ instr_filter }
      | assignation ';'         { instr_assignation }
      | funcall ';'                   { instr_funcall }
      | affect_operator ';'         { instr_affect_operator }
      | 'print' expr ';'          { instr_print }
      | 'print2file' '(' print_param ')' ';'    { instr_print2file }
      | conditional                 { instr_conditional }
      | loop_label? loop                  { instr_loop }
      | 'break' 'id'? ';'         { instr_break }
      | 'continue' 'id'? ';'           { instr_continue }
      | 'return' expr? ';'       { instr_return }
      | block                    { instr_block }
      | 'error' ';'               { instr_error }
      ;

print_param = 'id' 'comma' expr { print_parameters }
   ;
declaration = type 'id'                      { declaration_id }
| type 'id' '=' expr                        { declaration_id_init }
| type 'id' 'hat' '[' 'value_literal' ']'          { declaration_id_history }
| type 'id' 'hat' '[' 'value_literal' ']' expr { declaration_id_init_history }
| 'global' 'id' '=' expr               { declaration_global }
            ;

funcall = 'id' '(' arguments ')'               { funcall_id }
        | 'id' '.' 'id' '(' arguments ')'      { funcall_select }
        | primary '.' 'id' '(' arguments ')'   { funcall_primary }
        | 'id' '(' 'error' ')'                 { funcall_error_id }
        | 'id' '.' 'id' '(' 'error' ')'        { funcall_error_select }
        | primary '.' 'id' '(' 'error' ')'     { funcall_error_primary }
        ;
```

```
arguments = argument/'comma'*    { arguments }
          ;

argument = expr  { argument_expr }
         | '?'   { argument_none }
         ;

conditional = 'if' '(' expr ')' block                  { conditional_if }
            | 'if' '(' expr ')' block 'else' block     { conditional_if_else }

            | 'if' '(' 'error' ')' block               { conditional_error_if }
            | 'if' '(' 'error' ')' block 'else' block  { conditional_error_if_else }
            ;

loop = 'while' '(' expr ')' block                                  { loop_while }
     | 'do' block 'while' '(' expr ')'                            { loop_dowhile }
     | 'for' '(' for_loop_init? ';' expr? ';' for_loop_incr? ')' block
         { loop_for }
     | 'for' '(' 'id' 'in' expr ')' block                         { loop_foreach }

     | 'while' '(' 'error' ')' block                            { loop_error_while }
     | 'do' block 'while' '(' 'error' ')'                      { loop_error_dowhile }
     | 'for' '(' 'error' ')' block                               { loop_error_for }
     ;

filter_def = 'id' '.' 'filter' '(' expr ')'  { filter_declaration }
 ;



for_loop_init = declaration    { for_loop_init_declaration }
              | assignation    { for_loop_init_assignation }
              | funcall        { for_loop_init_funcall }
              ;

for_loop_incr = assignation     { for_loop_incr_assignation }
              | funcall         { for_loop_incr_funcall }
              ;

loop_label = 'id' ':'       { loop_label }
           ;

assignation = lhs '=' expr  { assignation }
            ;
```

```
lhs = 'id'                    { lhs_id }
    | field_access            { lhs_field_access }
    ;

primary  = field_access                                  { primary_field_access }
         | 'this'                                               { primary_this }
         | '(' expr ')'                                      { primary_parens }
         | 'id' '{' alloc_init* '}'                        { primary_allocation }
         | group_type                                           { primary_group }
         | list_type          { primary_list }
         | relation_type                                     { primary_relation }
         | funcall                                            { primary_funcall }
         | '(' 'error' ')'                                 { primary_error_parens }
         | 'id' '{' 'error' '}'                          { primary_error_allocation }
         | 'group' '[' 'error' ']'                          { primary_error_group }
         | 'list' '[' 'error' ']'           { primary_error_list }
         | 'id' '[' 'error' ']'                          { primary_error_relation }
         ;


alloc_init = 'id' '=' expr ';'     { alloc_init }
           ;

field_access = 'id' '.' 'id'        { field_access_id }
        | primary '.' 'id'     { field_access_primary }
        | 'id' '.' 'id' 'hat' '[' 'value_literal' ']' { field_access_history }
        | 'id' '.' 'id' 'hat' '[' 'value_literal' '..' 'value_literal' ']'
                                                  { field_access_history_range }
            ;

expr = 'boolean_literal'                                 { expr_boolean_literal }
     | 'value_literal'                                     { expr_value_literal }
     | 'id'                                                          { expr_id }
     | primary                                                    { expr_primary }
     | expr '..' expr                          [range]             { expr_range }
     | 'text_value'            { expr_text_value }
     | '!' expr                             [unary]             { expr_unary_not }
     | '+' expr                             [unary]            { expr_unary_plus }
     | '-' expr                             [unary]           { expr_unary_minus }
     | expr '==' expr                       [eq]                       { expr_eq }
     | expr '!=' expr                       [eq]                       { expr_ne }
     | expr '<' expr                        [eq]                       { expr_lt }
     | expr '<=' expr                       [eq]                       { expr_le }
     | expr '>' expr                        [eq]                       { expr_gt }
```

```
| expr '>=' expr                      [eq]                    { expr_ge }
| 'id' 'hat' '[' 'value_literal' ']'     expr_history }
| expr '&&' expr                        [boolean_and_or]  { expr_band }
| expr '||' expr                        [boolean_and_or]   { expr_bor }

| expr '+' expr                          [plus_minus]     { expr_plus }
| expr '-' expr                          [plus_minus]    { expr_minus }

| expr '*' expr                          [star_slash]     { expr_star }
| expr '/' expr                          [star_slash]    { expr_slash }
| expr '%' expr                          [star_slash]      { expr_mod }
;
```

Résumé

Les sciences qui traitent de la réalité, qu'elles soient naturelles, de la société ou de la vie, fonctionnent avec des modèles. Une partie de ces modèles décrivent les relations entre certaines grandeurs mesurables de la réalité, sans aller jusqu'au détail des interactions entre les éléments qui la composent. D'autres modèles décrivent ces interactions en prenant le point de vue des individus qui constituent le système, le comportement global n'est alors plus décrit à priori, mais observé à posteriori. Nous faisons le constat que dans les deux cas le scientifique a peu de liberté pour décrire les structures, en particulier spatiales, susceptibles de porter ces interactions. Nous proposons une approche de modélisation que l'on peut situer à mi-chemin entre les deux, et qui incite à étudier un système à travers la nature de ses interactions et des structures de graphes qui peuvent les porter. En plaçant au même niveau les relations spatiales, fonctionnelles, sociales ou hiérarchiques, nous tentons aussi de nous affranchir des contraintes induites par le choix effectué souvent à priori d'une forme de représentation de l'espace. Nous avons formalisé les concepts de base de cette approche, et ceux-ci ont constitué les éléments d'un langage métier, nommé Ocelet, que nous avons défini. Les outils permettant la mise en oeuvre de ce langage ont été développés et intégrés sous la forme d'un environnement de modélisation et de simulation. Enfin nous avons pu expérimenter notre nouvelle approche de modélisation et le langage Ocelet à travers la réalisation de plusieurs modèles présentant des situations variées de dynamiques paysagères.

Summary

Sciences dealing with reality, be it related to nature, society or life, use models. Some of these models describe the relations that exist between measurable properties of that reality, without detailing the interactions between its components. Other models describe those interactions from the point of view of the individuals that form the system, in which case the overall behaviour is not defined a priori but observed a posteriori. In both cases, it can be noted that the scientist is often limited in its capacity to describe the structures, especially those spatial, which support the interactions. We propose a modelling approach that can be considered intermediate, where the system is studied by examining the nature of the interactions involved and the graph structures needed to support them. By unifying the description of spatial, functional, social or hierarchical relationships, we attempt to lift constraints induced by the form of spatial representation that are often chosen a priori. The basic concepts of this approach have been formalised, and were used to define and build a domain specific language, called Ocelet. The tools related to the implementation of the language have also been developed and assembled into an integrated modelling and simulation environment. It was then possible to experiment our new modelling approach and the Ocelet language by developing models for a variety of dynamic landscapes situations.