Conception d'un simulateur multi-agents parcelle-troupeau

François Guerry, René Baumont, Bertrand Dumont, Laurent Pérochon

INSTITUT SUPERIEUR D'INFORMATIQUE DE MODELISATION ET DE LEURS APPLICATIONS

COMPLEXE DES CEZEAUX
BP 125 – 63170 AUBIERE CEDEX

INSTITUT NATIONAL DE LA RECHERCHE AGRONOMIQUE

UNITE DE RECHERCHE SUR LES HERBIVORES
THEIX
63122 ST-GENÈS-CHAMPANELLE

Rapport de stage 2eme année

Conception d'un simulateur multi-agents parcelle-troupeau

Tome I

Présenté par : François Guerry

Lieu de Stage :

Responsables du Stage : René Baumont,
Bertrand Dumont
Laurent Pérochon

Du 1er Avril au 30 Septembre 2000

INSTITUT SUPERIEUR D'INFORMATIQUE DE MODELISATION ET DE LEURS APPLICATIONS

COMPLEXE DES CEZEAUX
BP 125 – 63170 AUBIERE CEDEX

INSTITUT NATIONAL DE LA RECHERCHE AGRONOMIQUE

UNITE DE RECHERCHE SUR LES HERBIVORES
THEIX
63122 ST-GENÈS-CHAMPANELLE

Rapport de stage 2eme année

Conception d'un simulateur multi-agents parcelle-troupeau

Tome I

Présenté par : François Guerry

Lieu de Stage :

Responsables du Stage : René Baumont,
Bertrand Dumont
Laurent Pérochon

Du 1er Avril au 30 Septembre 2000

Remerciements

Je tiens à remercier M René Baumont et M Bertrand Dumont pour leur aide et leur expertise dans la conception du modèle, M Laurent Pérochon pour son soutien et ses conseils, ainsi que M Pascal Carrere et M Jean François Soussana pour leurs travaux simultanés.

Je tiens aussi à remercier toute l’équipe RAP pour son accueil chaleureux sur le site de l’INRA Clermont-Theix.

Table des figures et illustrations

Résumé

Ce stage consiste à l'élaboration d'un simulateur complexe d'un système parcelle/troupeau, visant à comprendre l'impact à long terme de l'activité de pâturage sur l'évolution de la végétation. Le travail réalisé se découpe en trois parties :

Premièrement, la construction d'un modèle général du comportement animal sur parcelle, sur la base de modèles préexistants et d'expertises variées. Deuxièmement, une modélisation plus détaillée d'une partie du modèle général. Ces modélisations ont été réalisées selon le processus UP, et grâce à la notation UML. Enfin, l'écriture en C++ du noyau de simulation à événements discrets, et l'implémentation de quelques classes du comportement de l'animal. Je me suis particulièrement attaché à réaliser un environnement de simulation stable, assez simple d'utilisation et souple.

Le simulateur, développé sous Linux, est à ce jour inachevé, et n'a donc pas pu être testé dans des conditions réalistes. Cependant quelques essais des possibilités du noyau donnent une idée des performances envisageables, qui me laissent optimiste.

Il reste bien entendu à terminer la programmation des classes, et à valider le modèle en procédant ci nécessaire à de petits ajustements laissés possibles dans l'architecture du logiciel.

Abstract

Table des Matières

Tome I :

Tome II : Annexes

Introduction

Dans le cadre d'un projet de recherche sur « l'utilisation par le pâturage et l'évolution de la végétation des surfaces herbagères hétérogènes et peu chargées », l'INRA s'est engagé dans la réalisation d'un simulateur informatique d'un modèle complexe parcelle/troupeau. Ce projet, au delà de l'échéance d'un stage, est appelé à mûrir encore beaucoup. En effet, la démarche scientifique impose que le modèle soit parfaitement validé pour être utilisable. Ce travail de validation est long et nécessite beaucoup de tests et d'ajustements du programme, et la conduite de manipulations de terrain susceptible de fournir des éléments de comparaison. Le stage se place dans l'optique de la conception du simulateur, à travers son analyse et son implémentation.

Mon rôle est donc de rassembler des modèles existants en un ensemble unique qui couvre les différents aspects du domaine. Le stage se situe plus particulièrement sur la partie animale du simulateur, prise en charge par l'équipe RAP, la partie végétale étant développée par l'équipe FGEP avec un autre stagiaire de l'ISIMA. Le problème est donc de réaliser un modèle du comportement animal prenant en compte de multiples facteurs, comme la motivation de l'animal, son état interne (digestion, notamment) et les interactions sociales au sein du troupeau,. Il n'est pas prévu sur la durée du stage d'arriver à un simulateur complet et opérationnel, mais seulement à une structure minimale (un noyau de simulation) et une première implémentation des fonctionnalités. En revanche, le souci principal sera d'analyser correctement le problème pour assurer une évolution facile et un meilleur soutien des modèles sous-jacents.

La démarche se décompose donc en trois phases, la première étant la modélisation conceptuelle du domaine ; puis la modélisation détaillée d'une sous partie du modèle général, concentrée sur l'activité de l'animal, à l'exception des relations sociales et des déplacements longs ; Enfin, l'implémentation du noyau de simulation et de ce sous modèle animal. Je vais donc commencer par vous présenter le contexte du projet, notamment le support de connaissances permettant l'élaboration du modèle, puis le déroulement de la conception du simulateur, et enfin je terminerais sur une étude du simulateur réalisé.

Chapitre 1 : Contexte scientifique et technique

1.1 Problématique agronomique

1.1.1 Objectifs du projet

Le projet de recherche sur l'utilisation des prairies hétérogènes et sous-chargées a pour objectif de comprendre le fonctionnement d'un système parcelle/troupeau en conditions extensives afin de pouvoir proposer des modes de gestion des prairies extensives permettant d'éviter leur dégradation et à terme leur embroussaillement. En effet, on constate aujourd'hui une dégradation des espaces ruraux, notamment sur les anciennes prairies de pâturage. Cela est dû à l'exode rural persistant dans la région, qui implique la réduction des effectifs au pâturage. Ainsi, on voit des prairies sous-exploitées ou abandonnées, qui se laissent gagner par une végétation broussailleuse. On peut citer par exemple les abords des gorges de la Monne, qui étaient autrefois des parcelles pour ruminants. Aujourd'hui, les pentes qui descendent aux gorges sont abandonnées, et la végétation ligneuse envahit l'espace. D'ici quelques dizaines d'années, on pourrait bien observer un changement notable des écosystèmes locaux.

Grâce à un mode de gestion adapté, il devrait être possible d'améliorer l'entretien des prairies extensives. Pour ces besoins, le projet AIP prévoit la conception et l'utilisation d'un simulateur informatique d'un modèle parcelle/troupeau, qui permettra d'étudier la dynamique des végétations de pâturage à long terme.

Un tel simulateur doit naturellement s'appuyer sur d'autres modèles déjà validés, et sur l'expertise des chercheurs. La conception elle-même est le point de départ des travaux de l'AIP ; l'objectif final étant de valider et d'utiliser le simulateur. Pour cela il faut des phases de tests, à mettre en parallèle avec des expérimentations et observations de terrain.

1.1.2 Objectif du stage

Le stage consiste plus précisément à concevoir un simulateur, basé sur une analyse biologique du problème. Cette analyse à été réalisée avant et pendant mon stage à partir des modèles existants déjà et grâce aux experts du domaine. Pour ma part, il a fallu réaliser un modèle informatique supportant ces concepts biologiques, par exemple la gestion du comportement des ruminants modélisés, et leurs interactions avec le milieu. Le travail consiste donc à analyser plusieurs modèles biologiques et à les adapter pour pouvoir les

inclure dans le programme. Ainsi, j'ai pu étudier des modèles de digestion, de comportement spatial et de relations sociales au sein d'un troupeau.

Mon analyse porte également sur la structure informatique du simulateur, et la manière dont il va accueillir le modèle parcelle/troupeau. J'ai ainsi pu mettre au point un noyau de simulation à événements discrets…. Toute l'analyse à été réalisée avec le processus UP, et le formalisme UML, auquel il a fallu familiariser mes interlocuteurs. Enfin, le stage prévoit une partie d'implémentation du simulateur qui ne vise pas à être terminée, mais au moins à tester l'utilisation d'une partie du modèle.

L'objectif principal du stage est ainsi d'arriver au terme de l'implémentation d'une partie du comportement animal. Cela nécessite que la phase d'analyse soit terminée et correcte, et que le noyau soit programmé et utilisable. L'implémentation consiste à rendre fonctionnel au moins un animal, de manière autonome. C'est à dire, sans interactions sociales, ni activités nécessitant des déplacements longs. Cela couvre uniquement les choix internes de l'animal entre trois activités : ingestion, rumination et repos complet. La phase d'ingestion consiste en une série de prélèvements sur la parcelle, et dépend elle aussi d'une série de choix : choix d'une cellule végétale à défolier, choix d'une quantité prélevée, tri sur la cellule.

Pour mieux appréhender tout cela, donnons une description du simulateur, tel qu'on doit le concevoir.

1.2 Description générale du simulateur

Le simulateur doit représenter le comportement alimentaire d'un troupeau de ruminants sur une parcelle de végétation hétérogène. La simulation va se dérouler sur des périodes longues, comme plusieurs années, pour bien appréhender la dynamique à long terme. Le simulateur touchant plusieurs domaines, on le structure en trois modules :

La première entité à modéliser, c'est la parcelle. On prévoit qu'elle fasse quelques hectares, et elle va être découpée en petites cellules indépendantes pour simuler la croissance des végétaux. C'est le module V (végétation).

Ensuite, il faut modéliser un animal, c'est à dire ses différents comportements, et une gestion de son état interne. Il pourra notamment manger, boire, ruminer et reposer. Tout cela concerne le module A (animal), qui est dédié au fonctionnement interne de l'animal.

Enfin, le module S (spatial et social) concerne plus particulièrement la modélisation des relations et interactions entre plusieurs animaux. C'est lui qui va donc gérer les déplacements longs sur la parcelle, et les actions de troupeau comme la grégarité (cohésion sociale) et la dominance de certains animaux pour le choix des lieux fréquentés (leadership).

Le simulateur sera conçu selon une approche multi-agents, qui permet à chaque individu biologique d'avoir un comportement propre uniquement relié aux autres par des interactions externes. La modélisation sera formulée grâce aux techniques de COO (UML), en utilisant le processus de conception itératif UP, qui permet une plus forte adéquation entre les programmeurs et les demandeurs. Il sera implémenté en C++ sur une plate forme de type Linux, avec les outils standards GNU : gcc (g++), la STL, et GNUMake.

1.3 Analyse biologique

1.3.1 Modèles biologiques préexistants

J'ai eu l'occasion d'étudier quelques modèles existants qui s'incluent dans le simulateur que l'on cherche à réaliser. Notamment, un certain nombre d'entre eux se rapportaient spécifiquement au module A, en ce qui concerne l'activité et l'état interne. Pour la plupart mécanistes, ils prévoient d'une manière souvent simple le comportement d'ingestion de l'animal en fonction de son état interne et des caractéristiques de l'aliment. Le modèle de Sauvant et al [1] est notamment le support d'un système de compartiments qui simulent la digestion, et prévoit l'ingestion à l'auge. La digestion est assurée par l'intégration d'un ensemble de compartiments représentant l'état du rumen. Le comportement est sélectionné parmi trois attitudes : ingestion, rumination et repos. On le détermine à partir des fonctions de motivation à ingérer, de satiété, de chargement du rumen, du bilan énergétique et de l'effet jour/nuit. Pour une vision plus détaillée du modèle, consulter l'annexe 1. La montre une représentation schématique du fonctionnement du modèle de Sauvant.

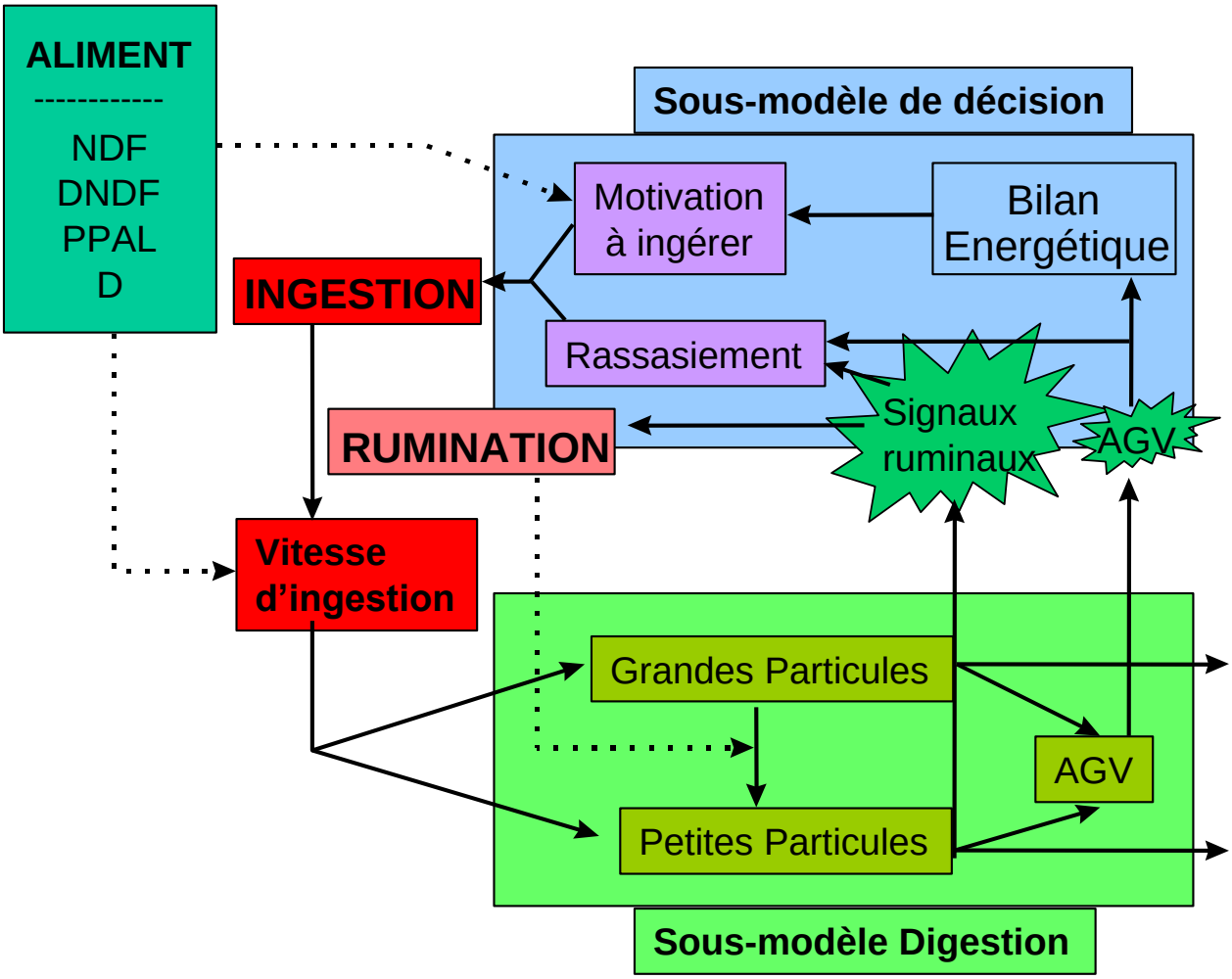


Figure 1 : diagramme conceptuel du modèle de Sauvant et al.

Le modèle d'ingestion à l'auge de Sauvant et al. a été adapté à une situation de pâturage sur une parcelle homogène [2]. Dans cette situation on considère que l'animal consomme le végétal par strates successives, ou horizons. Il s'agissait alors de modéliser les caractéristiques des différents horizons de pâturage et la vitesse d'ingestion qu'ils permettent pour l'animal, celui ci ayant alors à choisir entre les différents horizons disponibles. Dans ce modèle, le comportement n'est pas spatialisé puisque la parcelle est représentée par une superposition de strates homogènes. Le choix de l'animal se réduit au choix entre les deux horizons supérieurs disponibles. Ce modèle est schématisé par la Figure 2.

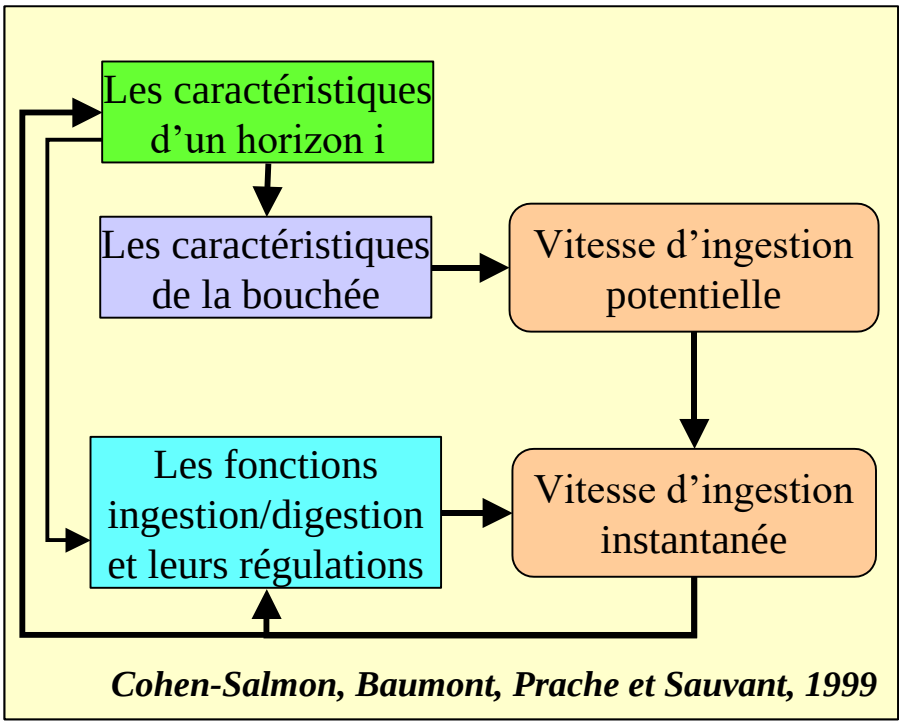


Figure 2 : schéma simplifié du modèle d'ingestion par horizons

On mesure alors le travail à accomplir pour passer à un modèle du pâturage sur couvert hétérogène. On va devoir envisager une approche ou l'animal aura à choisir entre des stations alimentaires variées, les cellules végétales.

J'ai aussi pu étudier des modèles se rapportant plus au module S, comme les travaux de B. Dumont sur le leadership [3] et la mémoire [4]. Le leadership est une loi de hiérarchie sociale du troupeau qui prévoit la dominance de certains individus sur le choix des lieux de pâturage et des activités. Autrement dit, certains animaux connaissant mieux la parcelle vont plus facilement imposer leur volonté de déplacement au reste du troupeau. Cette particularité s'applique dans le cadre des déplacement vers une zone de pâturage, pour l'activité alimentaire. Au sein du troupeau, il existe une répartition des possibilités de leadership entre les animaux, certains étant plus souvent leaders que d'autres. On observe même une proportion d'animaux passifs, qui suivent toujours et ne décident jamais, et une partie d'indépendants, qui agissent à leur guise sans entraîner le troupeau.

Le travail sur la simulation de la mémoire spatiale du mouton dans ces choix alimentaires permet de comprendre le rôle et le fonctionnement d'un modèle de mémoire des animaux. La mémoire sert aux animaux pour retrouver des sites alimentaires préférées, en l'occurrence il s'agissait de bols de concentré alimentaire dans le modèle de Dumont et Hill [4]. L'animal était lâché sur une parcelle contenant une certaine répartition spatiale de ces bols. La Figure 3 montre la structure du comportement de l'animal :

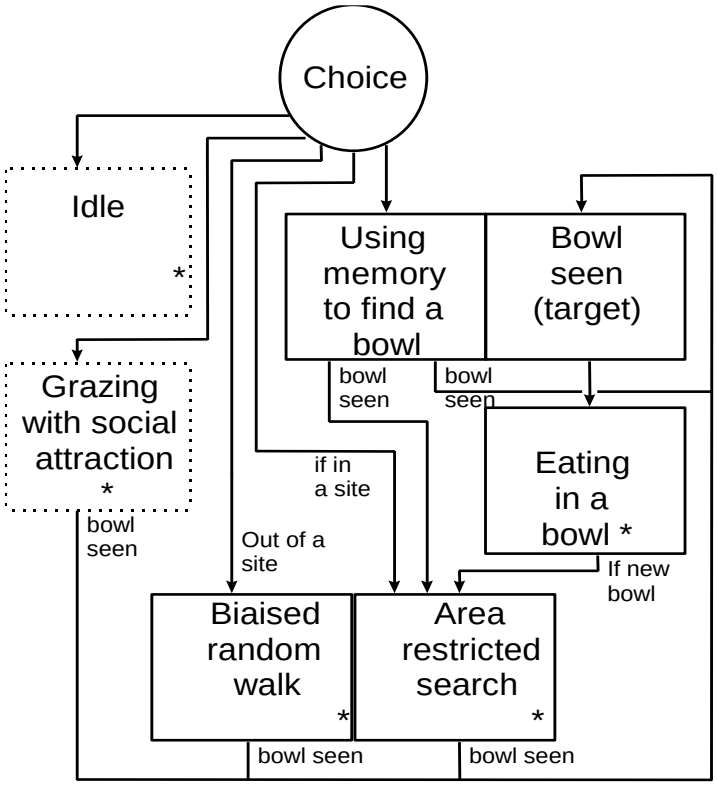


Figure 3 : diagramme des états d'un animal du modèle de mémoire spatiale

Le but est pour l'animal d'optimiser son alimentation, et la mémoire va donc contenir des informations sur les sites préférés, c'est à dire les plus intéressants. Il aura donc des aptitudes pour repérer les bols, et parcourir la parcelle pour les trouver. Le modèle de Bailey [5] complète cette approche, il simule des périodes plus longues, en gardant aussi en mémoire les sites à éviter. Ce modèle propose aussi des fonctions d'atténuation de la mémoire, qui simulent l'oubli, ou l'erreur.

1.3.2 Analyse des trois modules

a) Module V

Le simulateur utilise donc une approche spatialisée, et le module V est chargé de la gestion de la parcelle et des cellules végétales qui sont justement le support spatial. Pour permettre cela, il faut ainsi représenter d'un manière particulière les cellules, dans le but de définir des voisinages, et des distances. Chaque cellule va ainsi représenter une surface d'un dixième de mètre-carré, ce qui correspond à une station alimentaire typique (surface sur laquelle l'animal peut brouter sans bouger les pattes). On a retenu le choix d'un pavage de cellules hexagonales, déjà utilisé dans le modèle MODHETE de l'équipe FGEP, ce qui permet d'utiliser les six cellules voisines équidistantes, au lieu de huit cellules non équidistantes sur un modèle carré.

La parcelle se découpe selon deux schéma structurels : le premier est le plus général, il partage la parcelle en faciès de végétation. Un faciès correspond à une communauté végétale (une ou plusieurs espèces). Toutes les cellules d'un même faciès ne sont pas au même niveau de croissance. Un faciès va en fait déterminer un certain nombre de paramètres comportementaux des végétaux. Un faciès peut donc contenir un ensemble de cellules qui ne sont pas forcément voisines.

Le second découpage est celui des sites végétaux : un site est simplement un ensemble connexe de cellules appartenant à un même faciès. Le découpage en sites est utiles pour la gestion de la mémoire des animaux, qui gardera une image de ces sites.

D'autre part, la cellule possède une structure particulière, elle est subdivisée en quatre compartiments : le Végétatif Vert, le Végétatif Sec, le Reproducteur Vert et le Reproducteur Sec. Ces compartiments modélisent les différents états du végétal, jeune ou vieux, frais ou fané… Pour l'évolution d'une cellule, on utilise des fonctions générales qui modifient les compartiments : fonction Croissance, qui ajoute de la biomasse au compartiment VV ; fonction Sénescence, qui fait passer la biomasse des compartiments Verts aux compartiments Secs ; fonction Montaison, qui fait passer la biomasse du compartiment VV au compartiment RV (formation des tiges et organes reproducteurs) et enfin la fonction Litière, qui évacue les matières mortes des compartiments Secs. Par conséquent, la cellule végétale modélise une station alimentaire, mais ne fait pas de distinction sur les espèces qu'elle contient. En effet, une cellule est considérée comme homogène, l'hétérogénéité venant des variations entre différentes cellules.

b) Module A

Le module A tel qu'on le conçoit va reprendre pour une bonne part le modèle de Sauvant et al. [1], en y ajoutant de nouvelles fonctionnalités liées à la spatialisation du modèle. Notamment, on va étendre les comportements possibles : l'animal a le choix entre manger, boire, ruminer, repos court et repos long. La distinction entre ces deux derniers est assez simple : le repos long qualifie une période d'inactivité, générale au troupeau normalement, où l'animal se repose vraiment. Il va se coucher, dormir. Ces périodes apparaissent entre les repas, elles correspondent à la nuit et à la sieste en début d'après midi. Le repos court est une simple période d'inactivité ou d'indécision, un état temporaire où l'animal n'a rien envie de faire. La rumination est un cas particulier : ce n'est pas une activité à part entière, elle va se dérouler en parallèle à d'autres. En effet, l'animal peut ruminer dès qu'il n'avale rien, donc quand il ne mange pas et qu'il ne boit pas. Il s'agit donc des périodes de repos (long ou court), et éventuellement des phases de déplacement (voir paragraphe

suivant). A ce titre, le choix de ruminer n'est pas fait parmi les quatre alternatives de départ, qui sont hiérarchisées comme précisé sur la Figure 4.

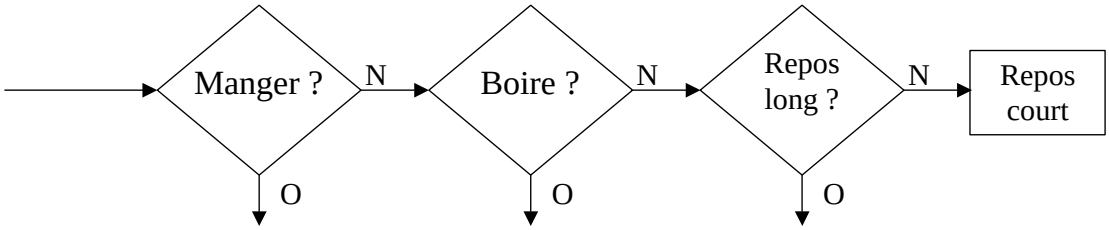


Figure 4 : hiérarchie des choix de l'animal

L'activité manger a priorité, puis boire, et enfin repos long. Le repos court est une attitude par défaut, en quelque sorte : absence de toute motivation. Pour chaque choix, il existe un critère particulier : pour manger, c'est celui du modèle de Sauvant, on compare les fonctions de motivation à ingérer et de rassasiement. Pour boire, on va se baser sur la comparaison de la quantité de matière sèche ingérée depuis la dernière fois ou l'animal a bu avec un seuil qui dépend de la température. Enfin, pour le repos long, on se base sur la motivation à ingérer qui doit être décroissante et suffisamment faible.

Pour la buvée et le repos long, on prévoit un (ou plusieurs) points particuliers sur la parcelle, des points d'eau, et des aires de repos. On va donc créer une liste de ces points particuliers, qui peuvent être uniques ou multiples (unique en tout cas dans un premier temps). Il seront considérés comme une connaissance globale et certaine de l'ensemble du troupeau. Pour pouvoir les utiliser, on va définir un « rayon d'accessibilité » dans lequel on considère que les animaux ont accès à l'activité associée au site.

Pour revenir sur la rumination, la décision peut donc se faire uniquement pendant les activités de repos ou de déplacement sur la base des caractéristiques du contenu digestif. Pendant ces périodes, on va donc lancer un cycle de rumination, qui fréquemment vérifiera si l'animal continue ou s'arrête de ruminer. Cela constitue un petit modèle à part, dont l'influence est de modifier temporairement les équations de digestion.

Je vais maintenant présenter l'interface entre les modules A et V, c'est à dire le modèle de défoliation des cellules. La défoliation comporte cinq sous parties : le choix d'une cellule à défolier, le temps passé sur la cellule, la quantité prélevée, le tri entre les quatre compartiments et la valeur nutritive du prélèvement. Le choix de la cellule se fait à partir de plusieurs variables : sa distance, sa direction par rapport à l'animal, et sa qualité nutritive. Cette dernière valeur peut être calculée de différentes façons : à partir d'un calcul empirique sur les hauteurs des compartiments, en fonction des préférences des animaux (relation obtenues par analyses de manipulations expérimentales), ou selon la vitesse d'ingestion permise par la cellule (selon l'Optimal Foraging Theory).

Le temps passé sur la cellule et la vitesse d'ingestion sont calculés en première approche à partir du compartiment VV. La quantité prélevée quant à elle, est simplement le produit du temps passé par la vitesse d'ingestion. Cette vitesse est modulée par la fonction de satiété de l'animal : s'il a moins faim, il mange moins vite. La répartition du prélèvement entre les quatre compartiments reflète le tri de l'animal : il va tenter de prendre plutôt les feuilles tendres et jeunes que de vieux épis secs… Pour cela, un modèle de tri à été mis au point, à partir de mesures de terrain et d'expertises. On a donc un modèle ovin, et un modèle bovin. Il prévoit un tri selon une maximisation du prélèvement sur le compartiment VV. Enfin, la valeur nutritive se calcule selon des critères de digestibilité et de proportions de parois végétales dans l'aliment. Ces critères vont êtres prévus grâce à des équations en correspondance avec l'état de développement du végétal. Pour de plus amples informations, l'ensemble des règles de défoliation sont présentées de manière détaillée dans l'annexe 2.

c) Module S

Le module S est chargé de la gestion spatiale et sociale des animaux. Il s'agit tout d'abord de prendre en compte les interactions du troupeau. Celles ci sont de deux types. Le premier type est la grégarité. En effet, les ruminants vivant en troupeau sont grégaires et se servent du troupeau comme d'une défense. Il vont donc rester groupés, pour mieux faire face aux menaces potentielles. On doit donc modéliser une attraction entre les animaux, ou plutôt une attraction de chaque animal vers le troupeau. On mettra donc dans le simulateurs des tests d'isolements et des fonctions pour se rapprocher du groupe.

Cela pose quelques problèmes, notamment pour définir ce qui représente le « centre » du troupeau, c'est à dire un point vers lequel un animal va se diriger. Evidemment, cela semble dépendre de la perception qu'a l'animal de ses congénères. On va donc premièrement choisir une condition de « sortie » du troupeau, puis déterminer une méthode pour le rejoindre. On considère qu'un animal est sorti du troupeau quand il ne voit pas au moins deux congénères à une distance inférieure à un seuil (à fixer). Un animal sorti du troupeau va le rejoindre, pour cela on dit qu'il se dirige vers le barycentre des congénères pondérés par leur proximité.

Le deuxième type d'interaction sociale est le leadership, dont on a parlé plus haut (cf. 1.3.1, modèle [3]). Plus précisément, on le modélise en donnant un attribut pour chaque animal, qui correspondra à une probabilité d'être leader. Ainsi, à chaque fois qu'un animal aura une volonté concernant le reste du troupeau, il devra tester par tirage aléatoire s'il peut ou non déclencher l'action. On considère de plus qu'il ne peut exister qu'un leader à la fois, et que tous les animaux finissent par le suivre… Le principe est simple : l'animal fait un choix d'activité (module A). C'est son désir. Pour réaliser ce désir, il doit vérifier si l'endroit est

adapté. Si oui, il exécute son désir. Sinon, il va devoir se déplacer. On fait un tirage aléatoire de leadership, en cas de succès, il part et avertit le troupeau. Sinon, il est contrarié et se rabat sur une activité par défaut, c'est à dire Manger si son désir était d'aller sur un site alimentaire éloigné, et Repos Court sinon.

L'utilisation de la mémoire est couplée. Celle ci va concerner uniquement le choix des lieux de pâturage, donc des cas de leadership vers un site alimentaire. L'animal va ainsi se rappeler des sites qu'il a déjà visité et pourra par la suite choisir entre eux, selon des critères de qualité et de proximité. Le critère de qualité va correspondre à la vitesse d'ingestion observée lors de la visite sur le site. La mémoire ne doit pas être infaillible, et s'efface peu à peu si elle n'est pas sollicitée. On reprendra directement le modèle de Dumont et Hill [4], auquel on va adjoindre certaines lois d'évolution, plus particulièrement décrites dans la publication de Bailey [5]. Elles concernent l'oubli et la mise à jour de la mémoire, selon lesquelles un site en mémoire qu'on ne revisite pas pendant un moment fini par être oublié, c'est à dire que le souvenir s'atténue, et la valeur du site se rapproche de la valeur moyenne de tous les sites. Quand elle est égale, l'animal a oublié, et on supprime le site de sa mémoire. En revanche, un site revisité sera mis à jour selon la nouvelle valeur observée, peut être modulée par le souvenir précédent.

Enfin, on va considérer que la mémoire n'entrera pas toujours en jeu : on peut laisser une pulsion exploratoire qui poussera l'animal à aller sur un lieu aléatoire. Cela arrivera de temps en temps lors des choix de sites, mais de manière plus fréquente si la mémoire est vide !

Je vais maintenant montrer un peu mieux le fonctionnement des déplacements de l'animal. Il est important de distinguer deux types de déplacements : les courts, qui concernent les quelques pas que fait un animal en broutant, ou en se rapprochant du troupeau, et les longs, qui correspondent à un changement de site. Le changement de site est en fait produit uniquement par leadership, et il peut être le fait de trois volontés : aller manger sur un autre site, aller boire, et aller se coucher. Cela correspond exactement aux activités Manger, Boire, et Repos Long. Au moment du leadership, l'animal leader va signaler au troupeau qu'il part vers un lieu précis et pour quelle raison. Les autres vont peu à peu le suivre et réaliser cette activité sur le nouveau lieu.

Le déplacement vers un site alimentaire est assez particulier, non seulement pour ses conditions de déclenchement, mais aussi pour le choix du site. Par opposition aux déplacements vers les aires de repos ou les point d'eau, où il suffit de se diriger vers un lieu connu, dès que la volonté s'en fait ressentir (cf. 1.3.2.b), le déplacement vers un site

alimentaire utilise la mémoire spatiale de l'animal. Il existe deux conditions pour déclencher ce déplacement :

1. la fin d'une période de repos long ou de buvée, c'est à dire le début d'une phase d'alimentation. Ce choix est clair et se produit après les activités en question.
2. la mauvaise qualité du site actuel, qu'on va remplacer par une meilleure alternative. Cela est plus compliqué, il faut définir un critère pour délaisser un site.

Le critère utilisé pour abandonner un site est basé sur la comparaison de la vitesse d'ingestion qu'il permet avec celle moyennes observée depuis les derniers jours. Si le site est en dessous de la moyenne, le troupeau va le délaisser pour un autre, choisi dans la mémoire spatiale. On va cependant prendre en compte une règle supplémentaire, qui spécifie que si sur le trajet, l'animal rencontre un lieu équivalent ou supérieur en qualité que celui qu'il vise, alors il s'y arrête. Cela permet de modéliser deux choses : premièrement, l'animal s'arrête dès son entrée sur le site, et ne file donc pas vers un point toujours le même, ce qui évite une répartition trop concentrée sur un point du site et quasiment inexistante ailleurs. Deuxièmement, un animal se dirigeant vers un bon site s'arrêtera selon toute logique s'il en trouve un meilleur, ce qui maximise encore plus son rapport qualité/déplacement.

1.3.3 Conclusion et questions en suspens

Finalement, les analyses précédentes ont permis de distinguer au cas par cas des sous-ensemble du modèle complet, en fournissant à chaque fois une petite partie du comportement à modéliser. Ainsi, on peut dire avec précision ce qui va se passer dans tel ou tel cas, mais il est encore délicat de dire comment va s'organiser le comportement général, c'est à dire l'enchaînement des cas particuliers.

Pour que cela soit plus clair, on propose de rassembler toute l'organisation sur un organnigramme algorithmique général, qui donne une vision d'ensemble du comportement de l'animal. Il est représenté par la Figure 5.

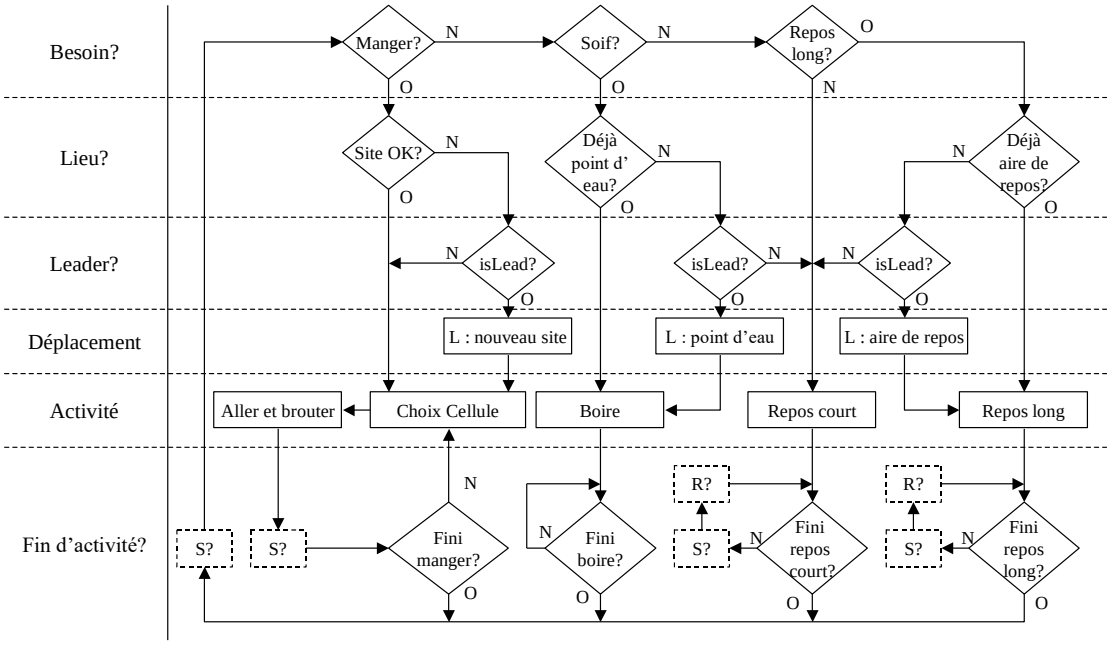


Figure 5 : organigramme algorithmique du comportement

On remarque la structure par couches logiques, qui permet de séparer chaque choix de comportement. Le besoin est exprimé en premier, puis on vérifie le lieu adapté à ce besoin. Si le lieu n'est pas adapté, on tente de devenir leader, et ont provoque éventuellement le déplacement. En fin de compte, on effectue l'activité retenue, et cela pour une certaine durée, pendant laquelle on peut s'occuper de la rumination et des interactions sociales, et on recommence.

Les boites R? et S? de ce diagramme symbolisent respectivement les cycles de rumination et les interactions sociale. Les déplacements ne sont pas explicités pour ne pas alourdir, mais contiennent aussi des cycles de rumination. Le contenu de ces boites est expliqué sur la Figure 6. L'organigramme général ne présente pas d'entée ou de sortie, on considère que l'animal effectue en boucle ses choix d'activité. En réalité, le simulateur entrera sur un choix, c'est à dire tout en haut. La sortie quand à elle n'est pas fixe, elle correspond simplement à l'instant où le temps de simulation est écoulé.

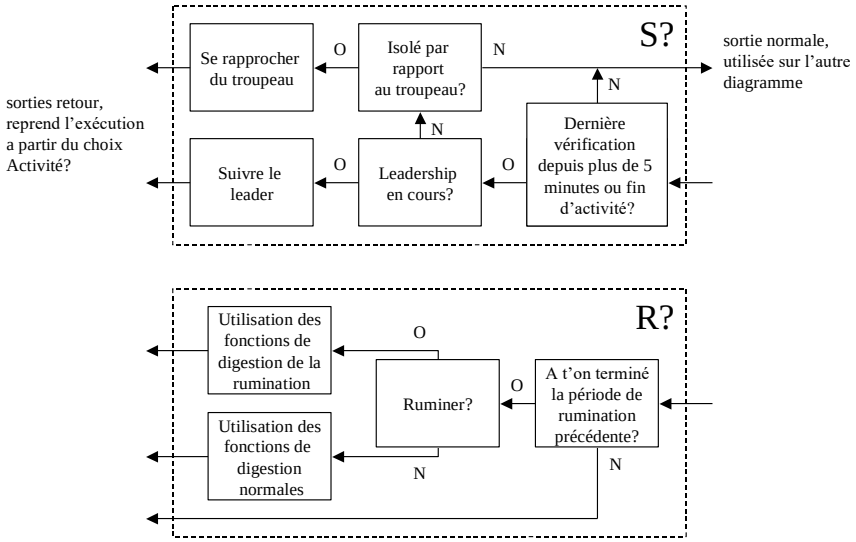


Figure 6 : contenu des boites de l'organigramme algorithmique

Pour le diagramme de la Figure 6, le test social contient deux choses, testées selon une priorité : d'abord la vérification du leadership. En effet, si un animal est devenu leader, le reste du troupeau va le suivre. C'est ici que cela est déterminé. Si un leader est en train de partir, naturellement on va le suivre. La seule exception concerne la boisson, qui ne peut être interrompue : l'animal ne suivra que lorsque il aura fini de boire, c'est pourquoi la boite S? est absente du cycle Boire de la Figure 5. S'il n'y a pas de leader, l'animal vérifie son éloignement au troupeau et le réintègre si nécessaire. Cette boite est vérifiée à des intervalles réguliers, mais pas en permanence, ce qui fait qu'on aura toujours un certain décalage entre un phénomène social et la réaction correspondante. Cela simule bien le fait que tout le troupeau ne part pas d'un bloc.

Pour la rumination, le cycle est effectué de temps en temps lui aussi, pour éviter que l'animal se mette à ruminer puis arrête sans cesse à haute fréquence…

Enfin, il reste une question qui n'est pas totalement résolue, concernant la représentation des sites alimentaires du point de vue des animaux. On pourrait dire qu'un site est vu comme tel qu'on le considère du point de vue végétal : un ensemble de cellules voisines appartenant à un même faciès. Mais l'animal ne connaît pas a priori le site entier, et logiquement, il se souviendra de l'ensemble des cellules qu'il a visité. On n'a pas précisé

plus que cela comment seront gérés les sites dans la mémoire de l'animal, ni les règles de perception exactes des sites.

Maintenant que le modèle biologique est suffisamment complet, il faut le transformer en une représentation informatique. Nous allons donc à présent voir les étapes de cette transformation…

Chapitre 2 : Conception du simulateur

Cette partie aborde la réalisation d'un simulateur multi-agents. Un simulateur multi-agents est une approche de la simulation où chaque élément intervenant dans le modèle du domaine est représenté de manière indépendante et unique. On les appelle alors des **agents**. Pour modéliser les comportements, on relie ces agents par des interactions, des communications…

La première étape consiste à transformer le modèle biologique en modèle multi-agents. Pour cela on s'appuie sur la conception orienté objet, et la notation UML. Je ne parlerais pas directement du processus de développement, dont la logique incrémentale est difficile à transcrire sur un rapport synthétique. Simplement, il faut savoir que le modèle UML se construit par étapes, en le soumettant aux expert, qui proposent des améliorations ou valident les modèles, autant de fois que nécessaire. C'est ce qu'on appelle le processus UP, qui conduit à définir peu à peu les besoins : les experts du domaine sont confrontés sans cesse à de nouvelles questions pour définir précisément quels sont ces besoins, et ce que doit faire exactement le simulateur.

Une deuxième étape pour construire le simulateur est de prévoir un modèle conforme à la représentation multi-agents, c'est à dire un noyau de simulation multi-agents. Le mien sera en outre construit selon une approche par événements discrets, comme on le verra.

Enfin, je présenterai une partie de l'implémentation du simulateur, notamment pour servir de guide de référence et d'utilisation pour le programmeur.

2.1 Analyse

2.1.1 Modélisation conceptuelle du domaine

Je vais maintenant expliquer la conception du modèle UML à partir du modèle biologique présenté dans le Chapitre 1. Je me parlerais ici du modèle général, et par la suite je passerais plus précisément sur le module A. Le modèle général à été représenté par un diagramme de classes simplifié, qui présente l'essentiel du problème. Il est montré par la Figure 7.

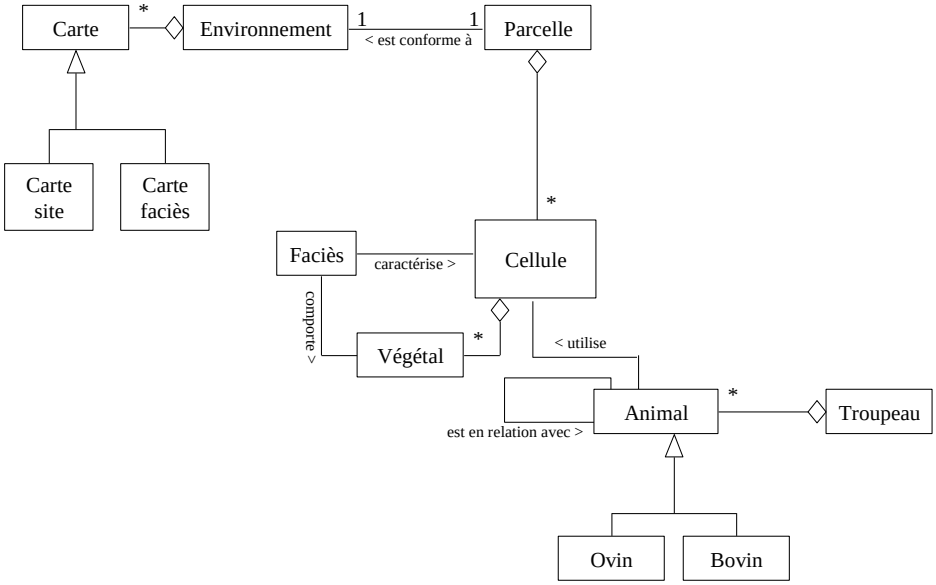


Figure 7 : diagramme de classes simplifié du domaine

Il comporte le minimum d'informations, à savoir qu'un animal appartient à un troupeau, et qu'il utilise des cellules. Les cellules sont conformes à des faciès de végétation qui contiennent des espèces végétales. Les cellules appartiennent à une parcelle, conformément à un environnement défini au départ par des cartes. L'animal est prévu pour être soit un ovin, soit un bovin. Ceci permet de situer un contexte minimal. Autour de l'animal, on va maintenant construire une structure qui comporte ses différents composants. On va rassembler les fonctionnalités par catégories, et les associer à des classes : on propose les catégories digestion, énergie et décision. La digestion va rassembler toutes les équations de digestion et leur mise en œuvre, la catégorie énergie s'occupe de tenir à jour les apports et besoins énergétiques et la catégorie décision va s'occuper des choix de l'animal.

La digestion telle qu'elle est présentée par les modèles biologiques est souvent assimilée au rumen, on utilise donc naturellement une classe Rumen pour représenter cela. On garde la possibilité d'inclure d'autres choses. On propose aussi une classe Centre Décisionnel pour les décisions. Certaines décisions vont faire appel à la mémoire, et de plus l'animal devra régulièrement mettre à jour cette mémoire. On propose donc en plus une classe Mémoire. L'intérêt de séparer tout cela est de simplifier les modifications futures. On saura directement dans quelle classe chercher pour toucher à tel ou tel comportement. En résumé, on obtient la structure agrégative présentée sur la Figure 8.

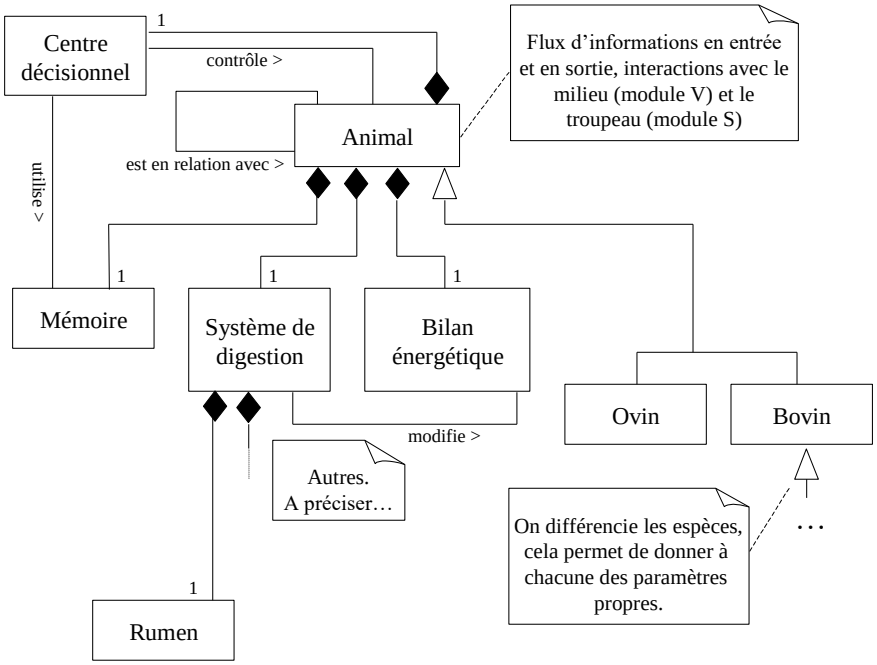


Figure 8 : diagramme de classes des composants de l'animal

Maintenant, on va rassembler dans un diagramme de cas d'utilisation les possibilités d'action qu'a un animal. Voici l'ensemble de ce qu'il peut faire, en vertu de notre extension du modèle de Sauvant : Manger, Boire, Repos Long (assimilable à dormir), Repos Court, et quelque chose pour la rumination. Les véritables actions comme Manger et Boire vont bien correspondre à des cas d'utilisation. Les repos aussi, mais ils sont différents seulement par leur durée : on ne garde qu'un scénario, qui contiendra cette variabilité : Repos. Certaines de ces actions nécessitent un déplacement avant d'être effectuées, comme Boire. On propose donc le cas d'utilisation Deplacement, qui représente les déplacements longs, correspondant au leadership. A ce propos, on mets donc un cas d'utilisation SuisJeLeader. Il existe aussi des déplacements courts, qui seront activés par l'activité Manger, mais ils font partie de l'ensemble Defoliation, qui d'un seul coup choisit une cellule, s'y rend et la défolie. Enfin, pour coordonner tout cela, il manque encore la possibilité de faire un choix : ChoixActivite, possibilité qui inclut celle de Digerer, et celle de Se Souvenir. La rumination est un comportement parallèle, on va se contenter de la déclencher et l'arrêter, cela influant sur la façon de Digerer. Au final, on obtient le diagramme présenté sur la Figure 9. Celle-ci présente les cas d'utilisation du point de vue du pseudo-acteur Animal.

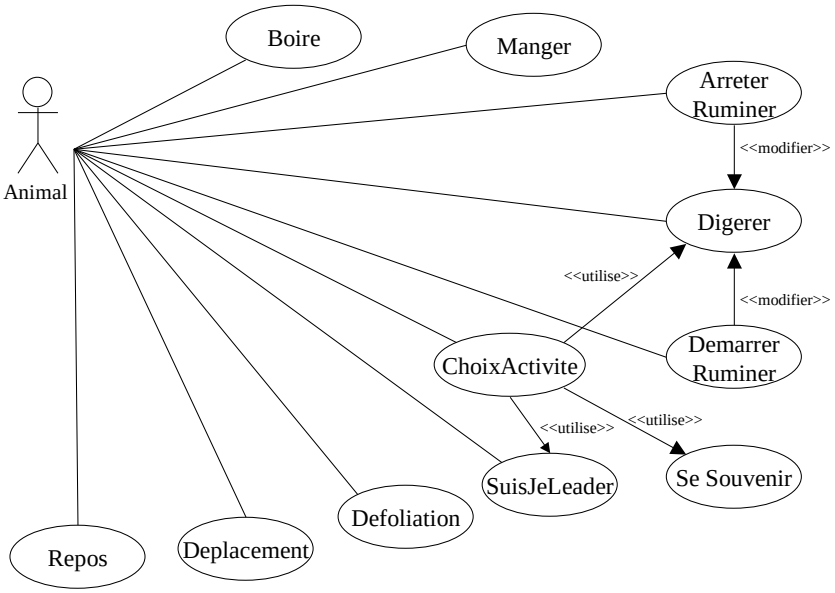


Figure 9 : diagramme des cas d'utilisation de l'animal

On complète enfin cette analyse par un diagramme d'état qui montre l'organisation des activités de l'animal. Il est présenté par la Figure 10.

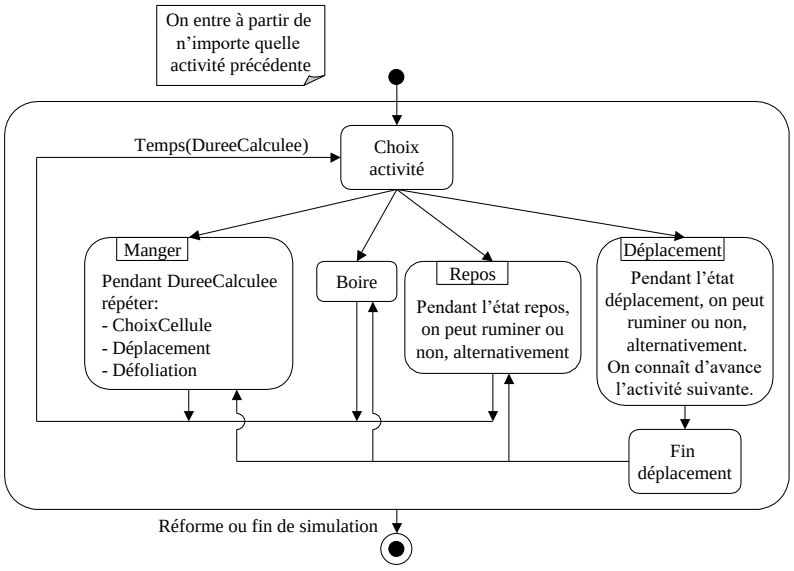


Figure 10 : diagramme d'états de l'animal

Ce diagramme montre comment se déroule chaque activité, en commençant toujours par un choix. Selon le choix, on execute une activité, qui est prévue pour une durée définie.

Après cette durée, on recommence. La seule exception concerne le déplacement, qui lui débouche non pas sur un choix, mais sur une autre activité directement : en effet, le déplacement (long) est le fait du leadership, et donc d'un choix d'activité particulière. En cas de déplacement, on connaît à l'avance quelle sera l'activité sur le site de destination. On remarque aussi les cycles de rumination pendant les états Repos et Déplacement.

A partir de cette modélisation générale, je vais maintenant concentrer l'étude sur le fonctionnement du module A. On va donc voir ses aspects plus dynamiques, grâce à d'autres diagrammes UML.

2.1.2 Modélisation conceptuelle détaillée du module Animal

Le modèle général de l'animal défini plus haut (Figure 8) représente les composants intuitifs du modèle. Une étude plus poussée va permettre d'approfondir certains aspects, de mettre en avant la nécessité d'autres classes, ou au contraire montrer que certaines sont inutiles. Notamment, on va procéder à une simplification : on a dit que le système digestif peut être assimilé au Rumen, cela suffit pour définir le comportement de l'animal dans le cadre de notre modèle. On supprime donc la classe intermédiaire « Système de Digestion ». De plus, on développe le rumen en différents compartiments, conformément au modèle adapté de Sauvant (Annexe 2). On prévoit la séparation des bilans énergétiques instantanés et cumulés, et les valeurs supplémenatires pour le calcul des certains comportements : les vitesses d'ingestions (VimC = sur le site en cours, VimL = sur les quatre derniers jours), et la QIDB (Quantité de matière sèche Ingérée depuis la Dernière Buvée). On a également besoin de lier le centre décisionnel à l'environnement (Milieu Physique) pour pouvoir connaître les éphémérides (fonction jour/nuit) et la température (pour la buvée). Le diagramme détaillé est sur la Figure 11.

Comme prévu, je me suis limité à l'étude approfondie du module A, qu'il est possible de faire fonctionner seul au prix de petites adaptations. En effet, il suffit de couper tout ce qui touche au module S, et laisser l'animal avec la seule possibilité de manger. Cela est judicieux, car la majeure partie du module A est déjà validée, comme le modèle de Sauvant. Ainsi seul le modèle de défoliation n'est pas validé, et des tests du modules A séparé devraient permettrent de faciliter sa validation.

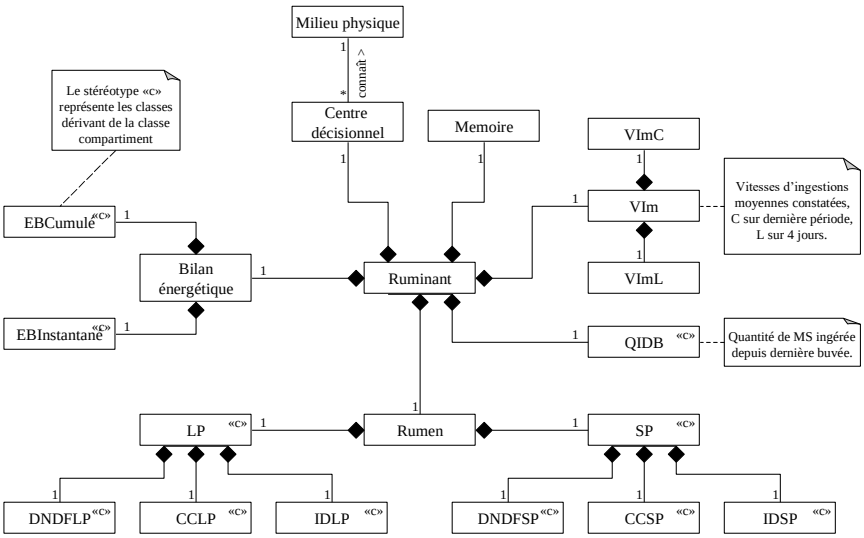


Figure 11 : diagramme de classes détaillé du module A

Le cœur du module A est le diagramme algorithmique (Figure 5), qui est indissociable du module S. On va donc tout modéliser, en sachant que toutes les sous-parties afférentes au module S seront ignorées. On va donc développer les scénarii du diagramme de cas d'utilisation (Figure 9), en commençant par le ChoixActivite. Ce choix, pour être mené à son terme, est particulièrement compliqué, puisqu'il reprend une bonne partie de l'algorithme structurel. Il est représenté sous forme hiérarchique par la Figure 12, qui montre les sous-unités qu'il utilise, et leur organisation. La première partie vérifie les interactions sociales, donc du module S, et renvoient au comportements appropriés si besoin est. On ne s'occupe pas de ça, notre modèle va pour l'instant passer directement sur PriseDeDecision. Celle ci correspond à la ligne Activite de la Figure 5, et détermine l'envie de l'animal. Selon cette envie, on vérifie ensuite le lieu ou doit se produire l'activité choisie, c'est le rôle des packages ChoixLieu[…]. Si le lieu est satisfaisant, on passe à l'activité elle même. Sinon, le leadership est testé, et on passe ensuite au comportement finalement nécessaire. On remarque le cas particulier du repos court, qui ne nécessite jamais de se poser une question de lieu. On passe directement, sans choisir de lieu. On le fait sur place.

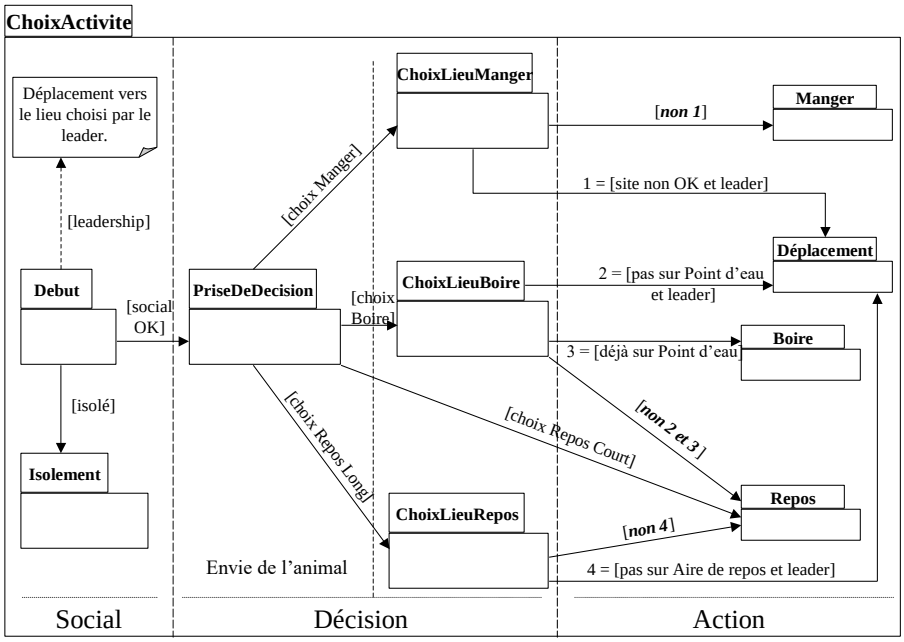


Figure 12 : diagramme de paquetages hiérarchiques du ChoixActivite

On va maintenant « zoomer » sur ces packages, et montrer le déroulement de chaque étape plus précisément. On ne précise pas le package Debut, de l'ordre du module S. On passe à PriseDeDecision, sur la Figure 13.

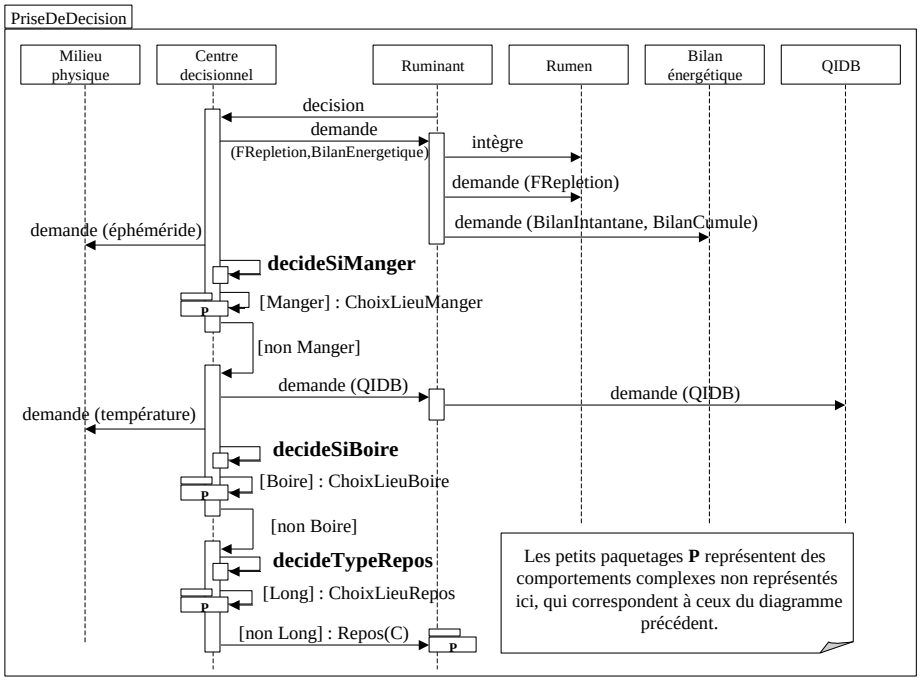


Figure 13 : diagramme de séquence de la prise de décision de l'animal

On observe sur ce diagramme l'organisation séquentielle du comportement : on appelle cela un diagramme de séquence pour cette raison. L'animal passe donc le contrôle

au centre décisionnel, qui doit collecter quelques informations avant de décider. Il consulte donc les entités nécessaires (animal, et milieu physique) et peut ainsi selon l'équation du modèle de Sauvant (cf. Annexe 2), décider s'il mange ou non. Ainsi de suite, il peut continuer sa collecte, et chaque terme en gras représente une équation qui permet de répondre oui ou non à un choix de comportement. Finalement, selon l'activité choisie, on passe le contrôle au package correspondant.

Pour les packages de choix de lieu, il est inutile de présenter les trois, puisque une bonne partie y est similaire. On présente seulement le plus complexe, pour le ChoixLieuManger (Figure 14).

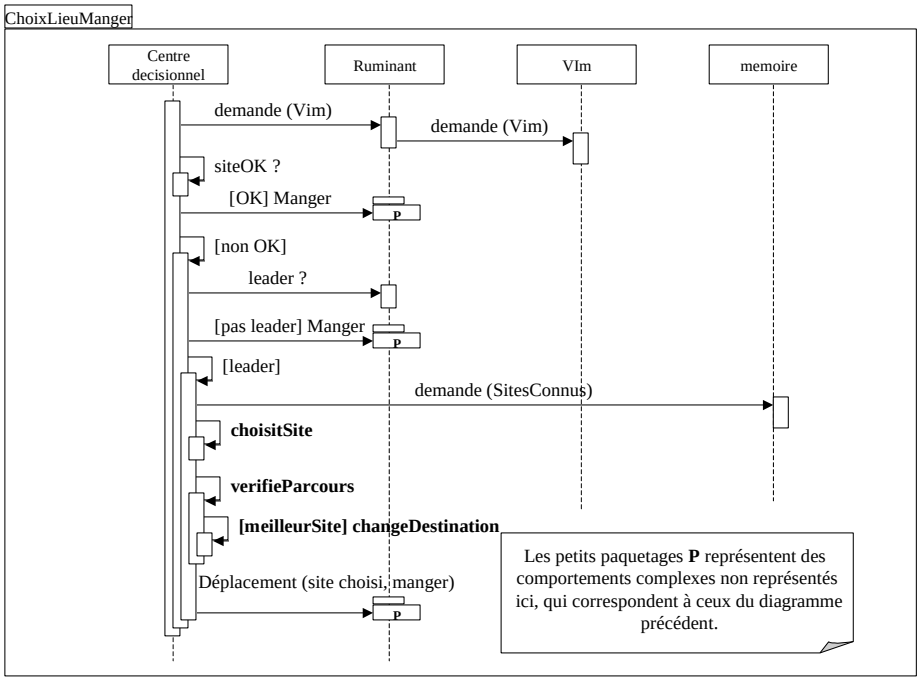


Figure 14 : diagramme de séquence du choix du lieu pour manger

On vérifie tout d'abord si le site convient, en demandant les valeurs des vitesses d'ingestions. Puis, s'il ne suffit pas, on teste le leadership pour déterminer si l'animal pourra ou non engager un déplacement. S'il peut, il cherche un meilleur site dans sa mémoire (module S), puis s'y rend. Evidemment, tout cela est plutôt orienté module S, mais il est nécessaire de prévoir dès maintenant cette structure pour éviter de tout avoir à refaire lors de la modélisation détaillée du module S. Enfin, je vais présenter un dernier diagramme, concernant l'activité Manger… Que se passe t'il exactement si l'animal mange ?? Voyons donc la Figure 15 :

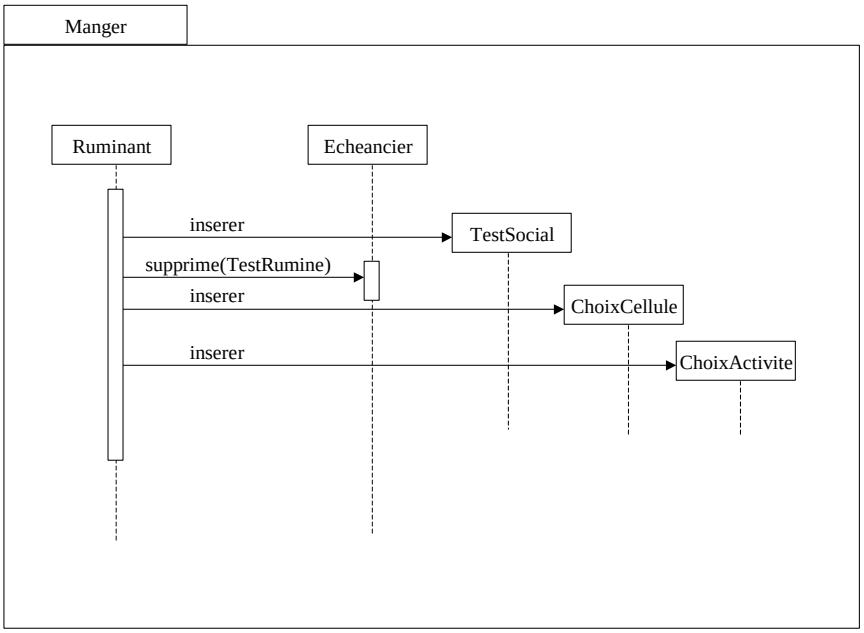


Figure 15 : diagramme de séquence de l'activité manger

A partir d'ici, on est obligé de définir un modèle d'enchaînement des comportements. On a retenu un modèle par événements discrets, c'est ce qu'on observe sur ce dernier diagramme : on insère des événements dans un échéancier… Maintenant que j'ai présenté quelques exemples du comportement précis du modèle, pour mieux comprendre ce principe d'événements, il est ici nécessaire de passer à la modélisation technique du simulateur…

2.1.3 Modélisation des classes techniques

Interfaces, echéancier, simulateur, packages Techno, Utils, Math. Sequence d'initialisation et déroulement.

On appelle noyau de simulation le moteur de toute la simulation : c'est lui qui supporte les agents du modèle. L'approche par événements discrets correspond à un mode de gestion du temps du simulateur. Cela revient à découper le temps selon les action des entités : chaque entité va donc être vu comme un objet subissant divers événements. Chaque événement va provoquer un comportement particulier de l'entité. Ainsi, lorsqu'on traite une activité, on va vouloir enchaîner sur d'autres. Pour cela, chaque activité va être déclenchée par un événement, qui lui même prévoira de nouveaux événements pour les activités futures.

Cependant, on peut se demander ce qui motive le choix de cette approche, par opposition à une horloge simple qui active les entités au bon moment. Pour répondre, il faut

considérer plusieurs choses : comment vont agir les entités ? quel est leur degré d'interaction ? quel est leur durée typique d'activation ?

Notre modèle biologique comporte de nombreuses entités, comme le troupeau, la parcelle, les ruminants, et d'innombrables cellules. Parmi ces entités, on constate que le temps entre deux activations peut être très varié : quelques secondes pour les ruminants, une journée pour les cellules… Gérer tout cela avec une horloge à pas fixe serait illusoire, puisque le pas de temps serait alors égal au temps minimal possible entre deux activations, c'est à dire de l'ordre de la seconde. Dans ces conditions, chaque entité serait vérifiée à chaque fois, mais activé que rarement, ce qui est loin d'être optimal. Grâce à l'approche par événements, le traitement se produit seulement si nécessaire, et les périodes où rien ne se passe sont sautées automatiquement.

La simulation par événements discrets est gérée par un échéancier, qui contient de manière chronologique les événements à traiter. Le principe de fonctionnement est simple : l'échéancier consulte le premier événement et active l'entité associée. En général, un événement donné va amener à insérer d'autres événements pour plus tard. Ainsi le traitement peut continuer indéfiniment, jusqu'à ce que le temps de simulation soit écoulé.

L'utilisation des événements et de l'échéancier est assez simple : au sens strict, un événement correspond à l'activation d'un comportement chez une entité particulière. On définit une entité comme étant un objet capable de gérer des événements. Dans cette modélisation, on utilise la classe Entite. Cette classe comporte l'interface minimale pour la gestion d'événements. L'échéancier contient une liste d'événements, chacun associé à une date d'activation. On utilise donc une classe CellEcheancier qui contient l'événement et la date associée. On utilise donc le modèle illustré par la Figure 16.

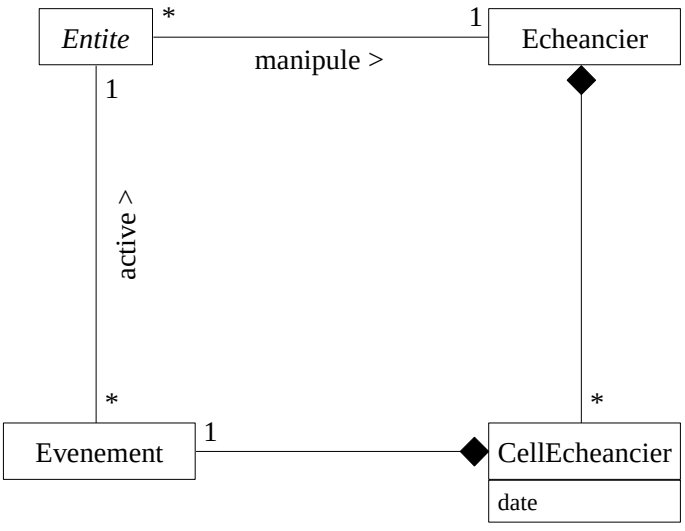


Figure 16 : diagramme de classes de gestion des événements

Au cours de la simulation, on suit l'échéancier, qui va exécuter un par un les événement. Pour cela, chaque événement est capable d'activer l'entité qu'il contrôle. L'activation va déclencher un traitement dépendant de l'entité et du type d'événement.

Pour reprendre l'exemple de la Figure 15, on voit en action ce principe. A l'issue d'une activité Manger (déclenchée par un événement, sans qu'on sache ici comment), on insère trois nouveaux événements : TestSocial, ChoixCellule et ChoixActivite. On supprime TestRumine. Tout cela signifie que pendant l'activité Manger, l'animal ne doit pas ruminer. Par contre, il va effectuer un choix de cellule, pour commencer à brouter, et il sera attentif aux interactions du troupeau (TestSocial). Enfin, la période pour Manger se terminera par un nouveau ChoixActivite ! Ce genre de modélisation permet de rendre compte de manière simple et indépendante du déroulement de chaque action. Consultez l'Annexe 3 pour une liste exhaustive des événements et de leur rôle.

Je vais maintenant présenter succinctement la classe Entite. Cette classe représente en fait les composants du simulateur : tout élément du modèle biologique devant évoluer et pouvant subir des événements en est un, il doit donc dériver de la classe Entite. Cette classe abstraite propose une interface générique d'utilisation avec des événements : elle contient le nécessaire pour manipuler un échéancier et des méthodes pour répondre aux événements. Elle peut aussi agréger d'autres entités, on parle alors d'un groupe (comme le Troupeau), et les liaisons entre conteneur/contenu sont prévues.

Pour représenter les modèles biologiques basés sur des modèles mathématiques, l'implémentation doit prendre en comptes les équations du problème. Notamment, on rencontre de nombreux modèles à compartiments. C'est pour cette raison qu'il existe une classe Compartiment, qui va simplifier la représentation des ces modèles, et automatiser leur traitement classique (intégration). Cette classe va permettre de relier les compartiments, et de les faire fonctionner en harmonie. Si on intègre tout un système de compartiments depuis une unique classe (comme Rumen), on peut ainsi cacher tout l'automatisme et n'avoir qu'a utiliser une fonction d'intégration sur la classe de plus haut niveau.

L'utilisation d'un simulateur n'a d'intérêt que s'il est possible de spécifier une situation initiale, et qu'on est capable de récupérer une sortie sur le déroulement de la simulation. Il faut donc naturellement prévoir des entrées/sorties. Dans le cas présent, le simulateur est suffisamment gros et coûteux en ressources pour qu'on ne s'encombre pas de gestion graphique pour l'instant. Les entrées/sorties se font donc grâce à des fichiers en mode texte. En fait, on va donner un fichier d'initialisation qui contient une définition de l'état initial du

simulateur : on fixe les paramètres, la valeur des variables au démarrage, l'état du système, les événements présents…

Le fichier est lu par la classe Lecteur, qui permet d'effectuer une interprétation du fichier : reconnaissance syntaxique et sémantique minimale, selon un formalisme simple que j'ai adopté. Cette classe fonctionne en parallèle avec Entite, l'initialisation du simulateur se fait grâce à leur interaction. En effet, la classe Lecteur va être utilisée tour à tour par chaque Entite nécessaire, qui va être ainsi initialisée. D'un autre coté, la classe Scribe va s'occuper de formater les sorties dans des fichiers reprenant le même formalisme. On peut donc dégager une structure synthétique comme présenté sur la Figure 17.

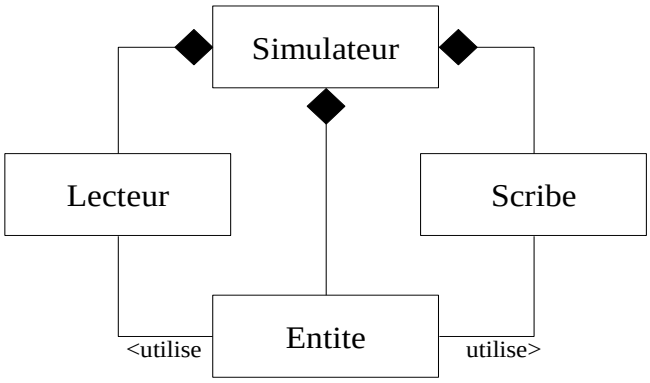


Figure 17 : diagramme de classe du système d'interface

Maintenant que l'on a vu l'agencement des différentes classes et leurs interactions, je propose de montrer comment se passe le déroulement de la simulation, à un bas niveau, c'est à dire depuis la gestion du noyau de simulation. En premier lieu, je présente tout de même le diagramme d'initialisation du simulateur (Figure 18).

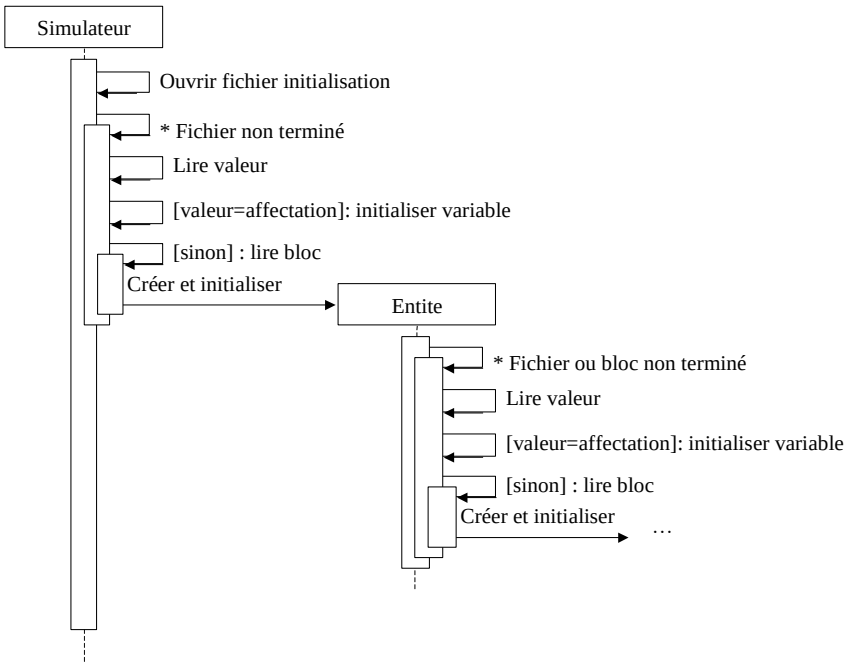


Figure 18 : diagramme de séquence d'initialisation

Ce diagramme est général, et construit des entités en les imbriquant, conformément au fichier d'initialisation. Un exemple concret serait la construction d'un troupeau, puis des ruminants qu'il contient. On montre maintenant le déroulement de la simulation :

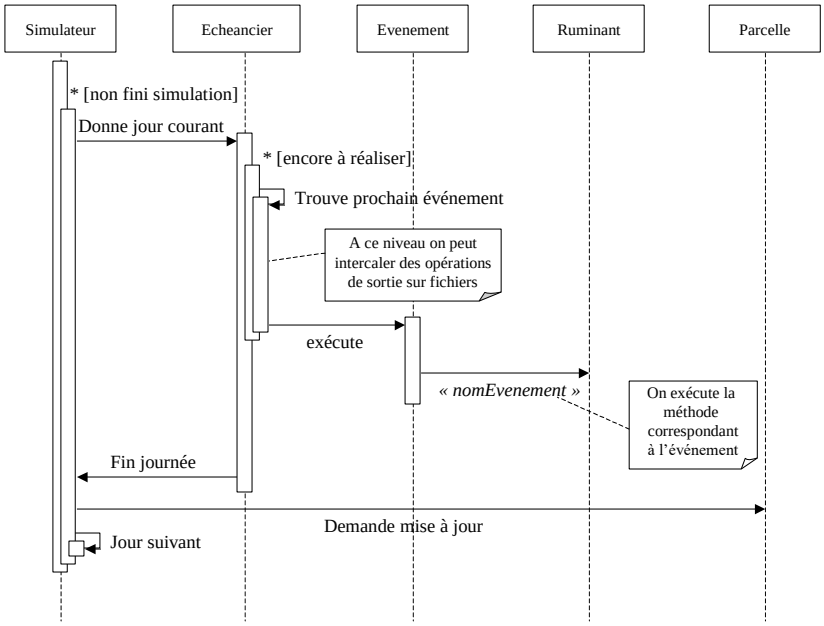


Figure 19 : diagramme de séquence du déroulement de la simulation

Chaque jour de simulation, on exécute tous les événements jusqu'au jour suivant. En dernier lieu, on effectue la mise à jour de la parcelle (croissance module V). L'exécution d'un événement passe le contrôle à l'entité qu'il active, pour effectuer le comportement approprié.

Toutes les classes techniques étant présentées, il faut maintenant songer à organiser le projet en sous unité logiques. On va donc diviser l'ensemble des classes en groupes de structure. On retient quatre groupe, c'est à dire les groupes Biolo, Techno, Utils et Math. On appelle cela de packages, dans le sens où ils vont contenir des composants du modèles UML. Dans le package biolo, on met toutes les classes bioligiques, c'est à dire les classes du domaine. Cela représente la partie comportement du simulateur. Le package techno referme les classes techniques du noyau de simulation : les événements, l'échéancier, le simulateur… Le package Math contient les classes dédiées au mathématiques, mais d'un point de vue général. Les équation particulières sont dans biolo, bien entendu. La package Math contient ainsi la classe Compartiment, et des générateurs aléatoires. Enfin, le package Utils propose de rassembler toutes les classes d'utilitaires dont on se sert et qui ne peuvent pas être classées ailleurs ; pour l'instant il y a ici les classes de lecture/écriture des fichiers, et une classe de gestion du temps (TimeUtil).

La présentation du modèle du simulateur, interfacé avec le modèle biologique, est maintenant terminée. Le simulateur modélisé est théoriquement viable et utilisable. On va donc se pencher maintenant sur l'implémentation explicite du programme de simulation.

2.2 Implémentation du modèle

L'implémentation du simulateur se fait donc en C++, langage à cheval sur la conception procédurale et la conception objet. Cela permet une souplesse dans les choix de développement, et une intégration optimisée. Cela reste un langage parmi les plus difficiles, car cette souplesse même donne au programmeur des possibilités infinies d'écriture, dont la plupart sont délicates à mettre au point. Néanmoins, au cours de cette partie, je vais présenter succinctement le mode d'emploi du simulateur, d'un point de vue programmation, dans le but de la poursuite du travail par un tiers.

Je vais donc présenter l'implémentation des diverses classes techniques du paragraphe 2.1.3, et leur utilisation dans le programme. Je présenterai aussi le modèle de mémoire utilisé pour optimiser les performances. Enfin, je préciserai aussi le fonctionnement précis de la gestion des entrées/sorties.

2.2.1 Interfaces des classes techniques

a) Classe Echeancier et événements

Toute entité peut manipuler l'échéancier : elle va notamment pouvoir insérer et détruire des événements qui la concernent. Elle comporte un attribut `ech` qui permet de manipuler l'échéancier, comme `ech->insere(...)` ou `ech->supprime(...)`. Plus précisément, la manipulation de l'échéancier se fait donc de la manière suivante, depuis une entité particulière :

Insertion d'un événement :

```
TimeUtil date = echeancier->getTime() + 60;
ech->insere(NomEvenement::newEvt(this),date);
```

On déclare un objet de manipulation du temps, qu'on initialise comme étant le temps courant plus 60. La méthode `getTime()` de l'échéancier retourne le temps courant, et l'opérateur + à été programmé comme un ajout de secondes au temps donné. Ensuite, on insère dans l'échéancier un nouvel événement, déclaré par l'utilisation des méthodes de gestion de mémoire décrites plus loin (cf. paragraphe 2.2.2). Il est inséré pour se déclencher à la date définie, c'est à dire dans 60 secondes.

Suppression d'un événement :

```
ech->supprime(NomEvenement::idC(),this);
```

Cela supprime toutes les occurrences futures de ce type d'événement pour l'entité en cours (`this`). La fonction `idC()` renvoie un identificateur qui permet de supprimer uniquement les événements de la bonne classe, et pas les autres.

b) Classe Entite

La classe Entite est une classe abstraite : elle ne sert que de point de départ d'une hiérarchie de dérivation, dont le comportement commun est d'être utilisé avec des événements. Elle peut être agrégée : en effet, la classe Entite contient un pointeur vers une autre entité, nommé `conteneur_`. Cela permet d'une part de ranger les entités dans une structure d'agrégation, et d'autre part d'accéder aux classes englobantes. Ainsi, par exemple un ruminant pourra connaître les informations concernant le troupeau.

Mais comment implémenter un modèle biologique avec cette classe ? En fait, quasiment chaque objet biologique devant avoir un comportement en rapport avec d'autres doit être une Entite. Par exemple, le troupeau, les ruminants, doivent en être. Par contre, la classe Rumen n'en est pas une, puisque elle est seulement agrégée sur le ruminant, et n'a pas de rapports directs avec des objets extérieurs au ruminant. Pour l'implémentation, il faut

prévoir ses cas d'utilisation et ses diagrammes de séquence : c'est beaucoup plus facile à programmer… On ecrit en général une méthode pas scénario d'utilisation, et cette méthode manipule l'échéancier. Enfin, il faut prévoir si nécessaire son initialisation : on écrit les méthodes de traitement des fichiers d'entrées (cf. 2.2.1.d).

c) Classe Compartiment

La classe Compartiment propose des attributs contenant la hauteur du compartiment depuis la dernière intégration, le pas de temps pour le calcul d'intégration `dt`, et le taux de variation du compartiment. En général, on décrit le modèle par des équations différentielle, c'est à dire de la forme : $\frac{\partial Compartiment}{\partial t} = Taux$. Le taux est donné par une équation. Pour faire fonctionner le compartiment, on va donc écrire cette équation dans la méthode qui donne le taux, appelée `delta()`. Par exemple :

```
float Rumen ::delta()
{
   return augment_-deduc_ ;
}
```

Où `augment_` et `reduc_` sont des attributs de la classe, entrant dans l'équation différentielle.

La méthode `integre()` se contente ensuite d'ajouter à la hauteur du compartiment la valeur nécessaire, soit `delta() * dt`. On peut néanmoins mettre en places des sous-compartiments, en les agrégeant, et en programment les méthodes d'intégration de manière imbriquée, on arrive à intégre tout le modèle de manière cohérente.

Par exemple, le Rumen contient des sous-compartiments SPcomp et LPcomp, et on veut intégrer harmonieusement. On réécrit integre :

```
void Rumen::integre (float dt)
{
   dt_ = dt ;
   spcomp_.integre(dt);
   lpcomp_.integre(dt);
   value_ = spcomp_.getVal() + lpcomp_.getVal();
}
```

Notons enfin que `integre` existe sous deux formes, l'une sans argument, que l'on appelle pour intégrer avec la valeur courante de `dt`, et l'autre avec un argument, qui donne la valeur de `dt`. C'est uniquement cette deuxième version qui a besoin d'être réécrite.

d) Classes Lecteur et Scribe

Les classes d'entrées/sorties permettent de lire et écrire des fichiers, dans des buts d'initialisation ou de trace de simulation. Lecteur lit et interprète un fichier pour le transcrire

en instructions d'initialisation, via des fonctions de la classe Entite. Scribe sort un fichier selon une représentation identique aux fichiers d'entrée. J'ai défini pour cela une petite syntaxe simple à respecter dans les fichiers d'entrée sortie. Je vais la présenter rapidement :

La syntaxe est basée sur des imbrications de blocs. Chaque bloc peut ainsi contenir d'autres blocs, ou des instructions d'affectation. La syntaxe reconnaît donc deux types d'instructions : les blocs, et les affectations :

```
<instruction> := [ '<bloc>' , '<affectation>' ]
```

Un bloc possède un type, et éventuellement un nom. On l'introduit par le mot clef `def:` et son contenu est mis entre accolades. On déclare donc un bloc par :

```
<bloc>  := 'def : <type_bloc> [(<nom_bloc>)]  { <instruction>* }'
```

le `<nom_bloc>` doit être entre parenthèses, les crochets signifient ici que c'est une partie facultative. L'étoile signifie qu'on peut mettre un nombre quelconque d'instructions à l'intérieur du bloc.

Une affectation permet de donner une valeur à un nom. Elle se termine par un point-virgule. La valeur est composée d'une séquence alphanumérique quelconque, mais sera interprétée comme un nombre si elle correspond à un format numérique et comme une chaîne de caractères sinon. On déclare une affectation par :

```
<affectation> := '<nom_parametre> = <valeur>;'
```

On peut à tout instant insérer des commentaires, pour cela on met un caractère # dans le texte. Ainsi, toute la fin de la ligne est sauté.

La classe Lecteur interprète et reconnaît automatiquement cette syntaxe, et renvoie des messages en cas d'erreurs. Les données analysées peuvent alors être fournies comme instructions d'initialisation pour le simulateur. Ainsi, toute la sémantique est définie par les classes qui utilisent Lecteur.

La sémantique est simple elle aussi : un bloc permet de définir l'initialisation d'une des entités agrégée. On part au début du bloc implicite « simulateur », et on peut atteindre les entités agrégées par des blocs : pour accéder au troupeau on fait dans le bloc principal :

```
def: troupeau (nomBidon)
{
   ...
}
```

Et ainsi de suite… il faut suivre la structure interne des agrégations pour imbriquer les blocs : si on déclare un bloc ruminant, il doit être dans le troupeau, pas dans le simulateur directement ! Comme je l'ai dit, l'analyse sémantique est faite directement dans les classes à initialiser. C'est le rôle de membres particuliers de la classe Entite. Il existe quatre méthodes pour l'analyse sémantique : une méthode générale qui parcours le fichier et lance les

comportements correspondants aux syntaxes rencontrées : c'est la méthode `init()`. Elle récupère la prochaine phrase du fichier, grâce à la classe Lecteur, et selon si c'est une affectation ou un bloc, elle délègue le traitement à des méthodes spécialisées : `goBloc()`, `declareReel()` et `declareChaine()`.

La première prépare l'extension à un nouveau bloc. Elle vérifie que le type de bloc demandé est disponible dans l'entité actuelle (par exemple : ruminant dans simulateur → NON, dans troupeau → OUI), puis délègue l'interprétation à la sous-entité agrégée qui doit être initialisée (en appelant son `init()`). Les deux autres méthodes vont vérifier que le paramètre existe dans le fichier, est bien compatible avec le type annoncé (réel ou chaîne) et le cas échéant, va affecter la valeur en question. Si un bloc ou une affectation n'a pas lieu d'être dans un bloc, une erreur va être lancée par une de ces trois méthodes. Ces trois méthodes doivent donc être reprogrammées dans chaque entité où c'est nécessaire, pour correspondre au contenu de celle-ci.

De manière symétrique, la classe d'écriture (Scribe) est prévue pour formater ses fichiers de manière conforme à la syntaxe. Elle utilise pour cela quatre méthodes : `openBloc(type ;nom)`, `closeBloc()`, `writeReel(reel)` et `writeChaine(chaine)`. L'utilisation est suffisamment intuitive, je dirais seulement que le fichier sera indenté automatiquement selon la structure des blocs.

Enfin, j'ai mis au point un petit utilitaire qui permet d'analyser et reformater des fichiers selon cette syntaxe. Ainsi après l'édition d'un fichier d'initialisation, il se peut que le texte soit mal aéré ou inexact. Le programme `reform` va signaler toute erreur, et indenter correctement chaque ligne en mettant une instruction par ligne.

2.2.2 Gestion de la mémoire

Pour des raisons de stabilité du programme, il a été nécessaire de développer des gestionnaires de mémoire pour certaines classes. En effet, le C++ est reconnu comme étant un langage performant et rapide, mais sa gestion de mémoire est parfois laborieuse et trop complexe, ce qui peut mener dans certains cas à des erreurs diverses qui plantent le programme. Cela arrive notamment lors d'utilisations très fréquentes de new/delete. Ainsi, la création de nombreux petits objets temporaires puis leur destruction peut entraîner des plantages, or nous avons besoin de ce genre de comportement : on va sans cesse créer et détruire des événements, par exemple… C'est pour ce genre de classes qu'il faut donc écrire des gestionnaires.

Le principe est assez simple : pour chaque classe de ce type, on va utiliser un tableau statique d'instances statiques, et pour qu'il puisse être très grand, on ne doit pas l'allouer sur

une instance dynamique, mais en statique. Pour cela, il y a deux possibilités : soit on déclare le tableau comme une variable globale (en dehors du main), soit on le met en attribut statique d'une classe. La deuxième solution est retenue et elle apporte une certaine dose d'originalité : on crée dans une classe un tableau statique d'instances statiques d'elle même en tant qu'attribut de classe, et on écrit un constructeur privé ! Personne ne peut instancier cette classe, et les seules instances disponibles sont dans le tableau.

On écrit donc une méthode de classe qui renvoie un pointeur sur une des instances, une libre. Cette méthode remplace le new habituel ! Ainsi, on ne peut accéder à cette classe que par des pointeurs, et on gère le tableau comme on veut : la méthode new prend la première instance libre du tableau, et la marque comme non libre. Une méthode de restitution est programmée aussi, qui remplace delete. Elle marque comme libre l'instance réstituée… Pour optimiser les acces aux instances libres/occupées, on gère en parallèle un second tableau qui contient entre deux index connus une liste contiguë de pointeurs sur les places libres. Quand on alloue, on donne le premier pointeur, et on avance d'une case dans le tableau. Quand on restitue, on avance l'index de fin, et la case libérée prend le pointeur sur l'instance restituée.

Cette technique à été utilisée pour tous les événement, et pour les cellules d'échéancier.

Chapitre 3 : Résultats et discussion

3.1 Résultats

3.1.1 Plate forme de travail

L'implémentation à été réalisée en C++, selon la norme stricte Ansi, avec le compilateur de GNU, g++ (version X.X.X). On a utilisé la version Linux, pour un meilleur contrôle et une stabilité renforcée. La machine de développement est un PC basé sur un processeur AMD Athlon, cadencé à 600Mhz, sur un bus à 100Mhz. Il possède 64Mo de mémoire vive. On a installé Linux Mandrake 7.0, version recommandée par l'INRA, et basée sur le noyau GNU 2.2.35. On a réservé un espace de swap de 768Mo sur le disque dur. Celui-ci est x.X.X.X.

Les tests sont effectués en mode console, avec un seul utilisateur loggé, sans autre processus. Cela permet d'obtenir des performances optimales mais utilise toutes les ressources de la machine.

Enfin, on utilise g++ accompagné de la STL pour Linux, et GNUmake pour gérer le projet.

3.1.2 Tests

3.2 Discussion

3.2.1 Déroulement du travail

Le découpage du temps de travail a suivi approximativement la logique de la modélisation, à savoir en trois parties principale. La première consistait à apprendre, comprendre puis étendre le modèle général. J'ai donc commencé par lire les articles de recherche concernant les modèles préexistant, comme le modèle de Sauvant [1] ou le modèle de mémoire [4]. J'ai lu d'autres articles, qui ne donnaient pas forcément un modèle complet, mais souvent des résultats pour modéliser les nouveaux comportements à prendre en compte (ingestion sur couvert hétérogène, tri, leadership, etc.). A partir de cette imprégnation, j'ai pu élaborer des ébauches du modèle, puis grâce aux interactions avec l'équipe RAP, celui ci a évolué jusqu'à atteindre l'état que j'ai présenté dans le chapitre 1. Cette évolution s'est déroulée par phases incrémentales de modélisation-validation, conformément au processus UP. Cependant, l'entendue des références bibliographiques ne

couvrait pas tous les détails de la modélisation. Plusieurs fois, les solutions retenues ont été proposées par expertise, ou parfois laissées en suspens… Cela a permis aux experts de se poser de nouvelles questions, auxquelles il faudra répondre pour valider le modèle. Cette première partie de modélisation a duré deux mois et demi, notamment pour permettre aux expert de réfléchir aux problèmes soulevés.

La seconde période, plus brève, correspond à la modélisation du noyau de simulation, et à l'intégration du modèle biologique dans celui-ci. C'est à ce moment que l'on s'est posé des problèmes sur la gestion du temps, des entités et de leurs actions. On a retenu l'approche par événements discrets, et j'ai donc adapté le modèle à cette approche. Pour cela, il a fallu spécifier la liste des événements à envisager, et l'action correspondant à chaque événement. On a aussi déterminé l'organisation générale du système, par exemple un traitement multi-agents, mais centré sur un unique échéancier (on sépare les agents biologiques et les objets techniques de simulation). Cette partie a aussi été contrôlée par les experts, qui devaient comprendre puis valider le nouveau modèle, sous une représentation dont ils avaient peu l'habitude. Cette partie a duré un mois, passé à mettre au point la structure et la coordination du simulateur.

Enfin, en dernier lieu, l'implémentation a commencé. Elle a été structurée autour des quatre packages de classes (biolo, techno, utils, math). La première étape a consisté à écrire les principaux objets techniques du noyau (échéancier, événements). Ensuite, j'ai programmé quelques entités biologiques ne comprenant aucun traitement, pour seulement faire tourner le noyau et vérifier le bon fonctionnement de l'échéancier et de la gestion de la mémoire. Puis j'ai programmé les classes d'entrées/sorties du simulateur, et écrit le petit outil de reformatage automatique. Enfin, je suis passé à une implémentation plus complète de quelques classes biologiques, comme le ruminant et ses composants. Cela a duré également un mois.

A l'issue de ces travaux, on peut faire tourner un simulateur comprenant de nombreuses entités, mais celles ci n'ont pas encore un comportement suffisamment avancé pour qu'on puisse parler de simulation d'un ruminant… Ensuite, je suis passé à la rédaction de ce rapport. D'ici la fin du stage (fin septembre 2000), j'espère avoir le temps de terminer l'implémentation du module A, et de le tester dans différentes conditions.

3.2.2 Assistance et contacts

Au cours de ce stage, j'ai eu la chance d'avoir des rapports avec de nombreuses personnes, dans les différents secteurs couverts par ce projet. Notamment, j'ai pu participer à deux réunions importantes sur le projet, qui rassemblaient l'ensemble des personnes

impliquées. Il y avait une partie de l'équipe RAP, une partie de l'équipe FGEP, des personnes pour le soutien informatique, et les deux stagiaires attachés au projet (Yannick et moi). Ces réunions de synthèse permettaient de présenter l'état du travail, et de définir la ligne de conduite pour la suite. De plus, on y a débattu sur un certain nombre de points clés, concernant particulièrement les interfaces entre les trois modules.

Ces réunions ont eu lieu le 5 mai, et le 29 juin. La première (au centre de Crouel) a permis de présenter les premières ébauches de modèles UML, et de lancer les débats sur la complexité des interfaces entre modules. La seconde (au centre de Theix) a relancé les problèmes non résolus pour ces interfaces, et j'ai pu y présenter les choix techniques retenus pour le simulateur : simulation par événements discrets, organisation des taches. De plus, une présentation plus détaillées des modules a été faite.

Au cours de ces réunions, et d'autre plus restreintes, j'ai pu bénéficier d'un soutien technique des professeurs de l'ISIMA. Notamment de Claude Mazel (tuteur su stage) et David Hill. Leur aide a été précieuse notamment pour le choix des solutions techniques. Le suivi technique n'a pas pu cependant être pris en charge par ces professeurs, et Laurent Pérochon s'est donc occupé de vérifier mon travail et de me guider pour la modélisation. Ils ont bien sur participé tous les trois aux réunions du 5 mai et du 29 juin, pour le soutien informatique.

Enfin, j'ai participé à de nombreuses entrevues avec les experts de l'équipe RAP, Bertrand Dumont et René Baumont. Souvent, ces réunions ont eu lieu avec Laurent Pérochon, et quelques fois avec Claude Mazel. Elles ont permis la séquence de réflexions, modélisations et ajustements qui ont conduit au modèle tel qu'il est aujourd'hui. Bertrand et René m'ont notamment aidé au cours de ces réunions à comprendre les modèle existants, et ont proposé des extensions à y apporter.

3.2.3 Avenir du projet

Actuellement, le simulateur est loin d'être terminé. La modélisation est presque complète, sauf en certains points, mais contient tant de nouveautés qu'il est possible que des pans entiers soient remis en cause. Cependant, l'objectif du stage est quasiment atteint : il s'agissait de modéliser et implémenter le module A. La modélisation de ce module est terminée, et a impliqué une bonne partie de celle du module S. L'implémentation comprend l'ensemble du noyau de simulation, et des fonctions d'entrées/sorties, mais seulement une

partie du module A. D'ici la fin du stage, je pense qu'il sera possible d'achever cette implémentation, et de tester en partie ce simulateur allégé.

Le projet va bien entendu continuer par la suite, et la partie technique (modélisation et développement) sera reprise par Laurent Pérochon, entré dans l'équipe RAP depuis juillet. En l'occurrence, il se pourrait que ma machine de développement soit utilisée comme serveur du projet, et que chaque protagoniste puisse y accéder à distance pour travailler dessus. Laurent va donc devoir terminer la modélisation du module S, et intégrer le module V de l'équipe FGEP. Il terminera le développement, et sera acteur de la validation du simulateur avec les experts du domaine.

Si nécessaire, bon nombre de modifications seront apportées au modèle, et des mises à jours permettront de la valider. Le but étant d'arriver à un système complet, si possible pour l'utiliser dans la conception d'un meta-modèle, plus rapide et totalement abstrait. Reste dans tout les cas à prévoir des outils (si possible graphiques) pour manipuler les entrées/sorties, et pourquoi pas le simulateur, en direct ?

Conclusion

Finalement, je suis arrivé à modéliser puis à implémenter (en C++) en partie le système parcelle/troupeau en conditions extensives sur couvert hétérogène. Le simulateur propose pour l'instant un fonctionnement minimal : initialisation des agents, et lancement du noyau temporel par insertion de quelques événements. Cependant, la majeur partie de ces événements ne sont toujours pas terminés, et le résultat est donc décevant : il ne se passe rien ! D'ici la fin du stage, le simulateur devrait comporter au moins le module A, conformément aux objectifs fixés.

Au cours ce cette conception, j'ai rencontré quelques difficultés, notamment pour la modélisation, ou parfois je me suis contenté d'une étude trop superficielle, et ne permettant pas de passer à une programmation aisée. Je peux citer aussi le casse-tête intellectuel qu'est le langage C++, particulièrement pour gérer certains concepts objet, comme les membres statiques.

Cependant, je pense que le stage m'a permis d'acquérir une nouvelle aisance sur ces domaines : je me représente beaucoup mieux maintenant la manière de mener une modélisation efficace, et je suis capable de programmer en C++ des ensembles de classes assez complexes…

Le projet est maintenant amené à être finalisé, pour prendre en compte les trois modules, et je pense qu'il est nécessaire de prévoir des outils intuitifs pour utiliser le simulateur. Bien entendu, pour que tout ceci soit utilisable, une phase de validation sera indispensable. Etant donné la nature complexe du simulateur, ce sera certainement long et difficile, et cela mettra en jeux de nouvelles expérimentations et observations à prévoir pour les équipes de l'INRA.

A terme, le simulateur pourrait être utilisé directement pour prévoir des dynamiques de végétation sur des données réelles. On pourrait aussi s'imaginer qu'un meta-modèle plus simple et plus performant permettrai de donner des résultats comparables, tout en étant plus souple. Le développement de ce genre de produit nécessite une validation poussée du simulateur, et un travail long d'abstraction.

Références bibliographiques

[1] D SAUVANT, R BAUMONT, P FAVERDIN, *Development of a Mechanistic Model of Intake and Chewing Activities of Sheep*, Journal of Animal Science, 74, 1996, pp 2785-2802

[2] D COHEN-SALMON, *Elaboration d'un modèle mécaniste intégrant l'architecture du couvert végétal et le comportement d'ingestion du mouton au pâturage*, mémoire de DAA sciences animales INAPG, 1999

[3] B DUMONT et A BOISSY, *Relations sociales et comportement alimentaire au pâturage*, INRA production animale, 12, 1999, pp 3-10

[4] B DUMONT et D R.C. HILL, *Multi-agent simulation of sheep spatial memory*, non publié. Disponible auprès de l'équipe RAP de l'INRA de Theix

[5] D W BAILEY, *Mechanisms that result in large herbivore grazing distribution patterns*, Journal of Range Management, 49, 1996, pp 386-400

INSTITUT SUPERIEUR D'INFORMATIQUE DE MODELISATION ET DE LEURS APPLICATIONS

COMPLEXE DES CEZEAUX
BP 125 – 63170 AUBIERE CEDEX

INSTITUT NATIONAL DE LA RECHERCHE AGRONOMIQUE

UNITE DE RECHERCHE SUR LES HERBIVORES
THEIX
63122 ST-GENÈS-CHAMPANELLE

Rapport de stage 2eme année

Conception d'un simulateur multi-agents parcelle-troupeau

Tome II

Annexes

Présenté par : François Guerry

Lieu de Stage :

Responsables du Stage : René Baumont,
Bertrand Dumont
Laurent Pérochon

Du 1er Avril au 30 Septembre 2000

INSTITUT SUPERIEUR D'INFORMATIQUE DE MODELISATION ET DE LEURS APPLICATIONS

COMPLEXE DES CEZEAUX
BP 125 – 63170 AUBIERE CEDEX

INSTITUT NATIONAL DE LA RECHERCHE AGRONOMIQUE

UNITE DE RECHERCHE SUR LES HERBIVORES
THEIX
63122 ST-GENÈS-CHAMPANELLE

Rapport de stage 2eme année

Conception d'un simulateur multi-agents parcelle-troupeau

Tome II

Annexes

Présenté par : François Guerry

Lieu de Stage :

Responsables du Stage : René Baumont,
Bertrand Dumont
Laurent Pérochon

Du 1er Avril au 30 Septembre 2000

Table des Matières

Tome I :

Tome II : Annexes

Table des Matières

Annexe 1 : Adaptation du modèle de Sauvant et al

Notations utilisées par le modèle de Sauvant

CC = contenu cellulaire (% de matière sèche)

CR = taux de transformation des LP en SP pendant la rumination

CVFA = concentration en acides gras volatiles du contenu ruminal (mmol/L)

D = coefficient d'hétérogénéité du fourrage $\in$]0;1[

DMM = matière sèche mastiquée effectivement pendant un cycle de rumination (g/min)

DN = effet Jour/Nuit

DNDF = digestibilité potentielle des parois cellulaires

EAT = variable pour Manger (0 ou 1)

EB = bilan énergétique instantané (balance entre le besoin et l'apport, 0 si tout est OK)

EBC = bilan énergétique cumulé de la veille

EC = rapport entre le besoin énergétique journalier et l'apport de la veille

FMI = fonction de motivation à ingérer

FSAT = fonction de satiété

IER = besoin énergétique instantané (kcal ME/kg $LW^{.75}$)

IDLP, IDSP = matière sèche non digestible dans les particules petites (S) et grandes (L)

IR = vitesse d'ingestion (g DM/min)

KA = taux maximal d'absorption d'acides gras (min^{-1})

KCC = taux de dégradation du contenu cellulaire (min^{-1})

KNDF = taux de dégradation des parois cellulaires (min^{-1})

KNDFM = valeur potentielle de KNDF (min^{-1})

LP = grandes particules (diam > 1 mm)

LPI = flux entrant de LP (g DM/min)

LPM, SPM = flux de dégradation microbienne des particules (S et P) (g DM/min)

LPR = flux de transformation des LP en SP pendant la rumination (g DM/min)

LPT, SPT = flux de transit des particules (S et P) (g DM/min)

LW = poids vif (kg)

MER = besoin énergétique quotidien (kcal ME/kg $LW^{.75)}$

MPS = taille moyenne des particules (mm)

NDF = taux de fibres dans la plante, méthode des détergents neutres (% DM)

PAL = palatabilité instantanée du régime

PIR = vitesse d'ingestion potentiel (g DM/min)

PLPF = proportion de particules larges dans le régime

PLPO = proportion de particules larges passant par l'ORO

PLPR = proportion de particules larges dans le rumen

PLPS = proportion de particules larges dans l'aliment mastiqué (0 < PLPS < 1)
PPAL = palatabilité potentielle
PR = probabilité de rumination
PROO = quantité de DM passant par l'orifice reticulo-omasal (g/min)
QIC = quantité ingérée cumulée (depuis dernière buvée)
R = proportion du fourrage restant par rapport au début (0< R <1)
RCM = nombre de sequnences A de contractions reticulo-omasales par minute
RDM = quantité de DM contenue dans le rumen (%)
RLS = signal de chargement du rumen
ROO = orifice réticulo-omasal (ORO en français)
RUM = variable pour ruminer (0 ou 1)
RV = volume du rumen (L)
SP = petites particules (diam < 1mm)
TD = seuil de décision pour l'activité d'alimentation
VDMI = ingestion volontaire de matière sèche (g DM/d)
VFA = acides gras volatiles (AGV), quantité totale dans le rumen
VFAF = production de VFA due à la digestion microbienne
VFAI = quantité de VFA déjà présente dans l'aliment (obtenu par ensilage)
VFAA = absorption de VFA pour les besoins énergétique de l'animal
VFAT = flux de transit de VFA, partie non consommée

Système d'équations utilisées par le système continu du module A

Ingestion: (toutes ces équations seront adaptées aux cas qui nous concernent)

PAL = PPAL * R^D adapte l'attrait potentiel du fourrage en fonction du souvenir R de l'animal

PIR = (16.8 - .14 * NDF * PLPF) * PAL ingestion potentielle dépendante des param. de la plante

IR = [PIR + 1.47 * ($LW^{.75}$ - $60^{.75}$)]/FSAT ingestion réelle dépendante des param. de l'animal

PLPS = 1.21 * (1 + 1.14/PLPF)$^{-1}$ effet de la mastication sur la qt. de LP

Rumen:

dLP/dt = LPI - LPR - LPT - LPM variation des réserves ruminales de LP

LPI = EAT * PLPS * IR flux entrant (vient de l'ingestion)

LPR = RUM * DMM * PLPR * CR flux de rumination

CR = .6 constante de transformation

DMM = 3.5 + (LW -60)/10 + (LP - 4 * LW)/100 dépend du gabarit de l'animal

PLPR = LP / (LP + SP) logique!!

LPT = RCM * PROO * PLPO sortie de matière du rumen

RCM = (1 + .1 * IR) * EAT + (1 - EAT) fréquence d'évacuation du contenu ruminal

PROO = .40 * RV / (150 * LW) qt évacuée par contraction

PLPO = .05 constante de sortie des LP

LPM = DNDFLP2M + CCLP2M digestion microbienne

dSP/dt = SPI + LPR - SPT - SPM idem LP, autres équations a déduire :

SPI = EAT * (1 - PLPS) * IR

SPT = RCM * PROO * (1 - PLPO)

SPM = DNDFSP2M + CCSP2M

dDNDFLP/dt = LPI * .01 * NDF * DNDF - DNDFLPR - DNDFLPT

- DNDFLPM

digestion LP NDF flux lag→digestion ... cf. sorties ci-dessous

DNDFLPR = LPR * DNDFLP/LP rumination → proportion de la valeur générale

DNDFLPT = LPT * DNDFLP/LP transit → idem

DNDFLPM = .8 * KNDF * DNDFLP digestion microbienne → qt * taux de digestion

KNDF = KNDFM * $(1+(.5/(pH - 5)))^{-1}$ dépend du pH

pH = 7.7 - .014 * CVFA acidité du milieu ruminal

KNDFM = 10 * $(1 + (NDFRum – 55)/10)^{-1}$ nouvelle équation, déterminée à partir des données expérimentales. On utilise :

NDFRum = 88 * (DNDFLP + DNDFSP + IDLP + IDSP)/(LP + SP)

dCCLP/dt = LPI * (1 - .01 * NDF) - CCLPR - CCLPT - CCLPM idem NDF

CCLPR = LPR * CCLP/LP idem

CCLPT = LPT * CCLP/lp idem

CCLPM = KCC * CCLP idem

KCC = 5 * KNDF voila l'explication

dDNDFSP/dt = SPI * .01 * NDF * DNDF + LPR * DNDFLP/LP + DNDFLPR - DNDFSPT - DNDFSPM

digestion SP NDF flux entrée, cette fois! sorties

DNDFSPT = SPT * DNDFSP/SP même principe…

DNDFSPM = KNDF * DNDFSP logique.

dCCSPT/dt = SPI * (1 - .01 * NDF) + LPR * CCLP/LP + CCLPR - CCSPT - CCSPM c'est pareil…

CCSPT = SPT * CCSP/SP

CCSPM = KCC * CCSP

IDLP = LP - DNDFLP - CCLP ce sont les restes….

IDSP = SP - DNDFSP - CCSP

Production énergétique :

dVFA/dt = VFAF + VFAI - VFAA - VFAT entrées/sorties pour VFA

VFAF = CVFA*RDM*(LPT+SPT)/1000 production par fermentation microbienne

VFAI = équation non pertinente pour les cas étudiés (= 0, pas d'ensilage)

VFAA = KA * VFA * (1 + (KVFA/CVFA))$^{-1}$ absorption de VFA

KA = .02 min^{-1}

KVFA = 100 mmol/L

CVFA = VFA/JUS on prend le volume total

JUS = RV-((LP+SP)/850)

VFAT = CVFA * (LPT +SPT)/(.01 * RDM) sortie par transit

Fonctions de décision :

FMI = EC * PAL * DN logique!

EC = max(MER/(MER+EBC) , 1) dépend de la veille

EBC = somme des EB de la veille.

EB = .288 * VFAA/.66 - IER énergie fournie – besoin

IER = MER * step / 1JOUR besoin = p/r à 1 journée.

FSAT = RLS * exp(.1 * EB/IER) fonction de satiété

RLS = exp(RV - .15 * LW)/(.15 * LW) chargement du rumen

RDM = 7.68 + .004 * (LP + SP)/.85 OK. C'est un état des choses

RV = .001 * (LP + SP) * (100/RDM + 1)

PR = PRMAX/[1 + (.15/PLPR)^2]

Annexe 2 : Analyse de l'interface entre les modules A et V

Synthèse des contributions écrites de R. Baumont et de P. Carrère à partir de discussions ayant associées B. Dumont, S. Prache, F. Louault, P. D'Hour, F. Guerry et L. Pérochon.

L'interface entre les modules A et V consiste dans le comportement de défoliation à l'échelle de la cellule de végétation. A partir des informations calculées par le module V il faut pouvoir définir les règles qui détermineront le choix des cellules défoliées par l'animal. Ensuite le simulateur doit pouvoir calculer le prélèvement réalisé sur chaque cellule en quantité et en nature.

1/ Choix de la cellule à défolier :

Il s'agit de classer les cellules présentes autour de l'animal avec un critère synthétique qui doit intégrer trois notions : la valeur alimentaire de la cellule, la distance et la direction.

1.1) Valeur alimentaire

Les principaux facteurs pouvant intervenir dans l'appréciation par l'animal de la valeur de la cellule sont : l'abondance de limbe vert, la hauteur (préfère le haut au ras), le ratio de tissus reproducteurs (notion de seuil), la densité (préférence du ras dense vs haut peu dense). On propose de tester trois fonctions de détermination de la « valeur alimentaire », deux sont une approche empirique, la troisième est issue de l'Optimal Foraging Théory.

Scénario Empirique 1 **: La qualité de la cellule est déterminée par un indice synthétique incluant le ratio de tissus reproducteurs, le ratio de tissus sénescents, la hauteur et la biomasse de vert**.

VALi = R * S * H * B

Avec R = (biomasse RV + biomasse RS) / (biomasse VV + biomasse VS)
S = (biomasse VS + biomasse RS) / (biomasse VV + biomasse RV)

H = Hauteur pondérée des compartiments verts

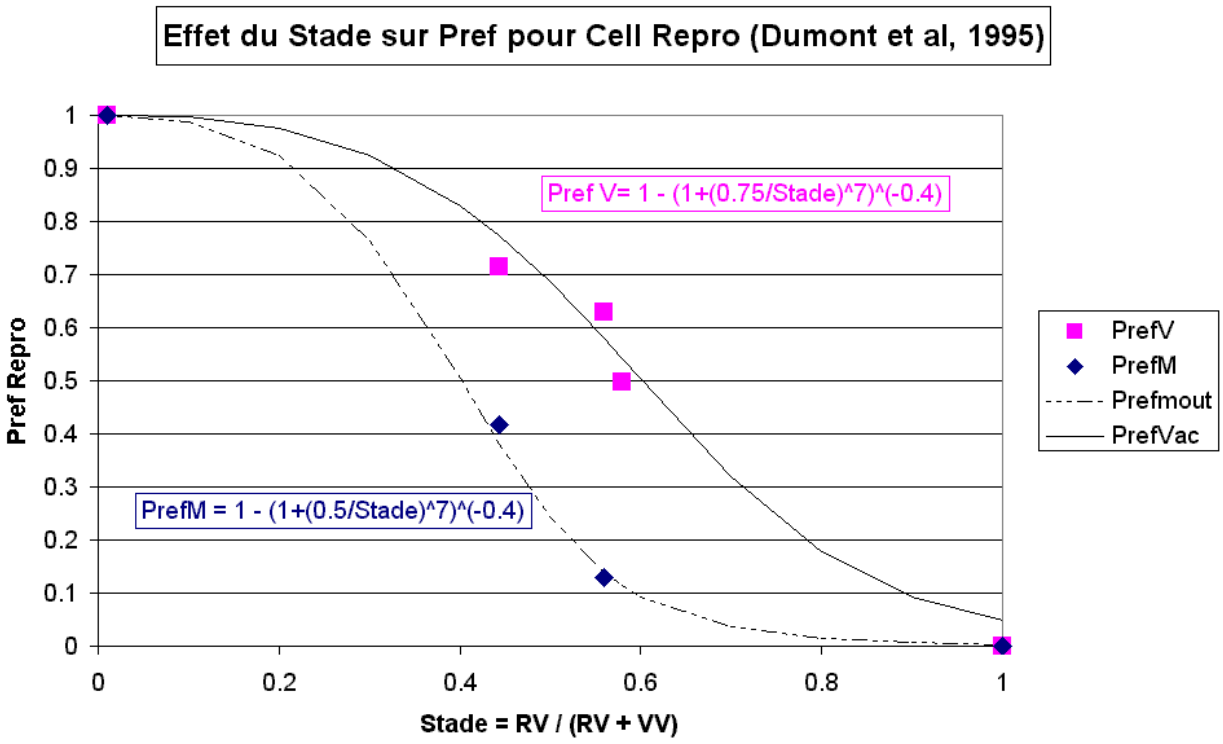
= ((biomasse VV * hauteur VV) + (biomasse RV * hauteur RV)) / (biomasse VV + biomasse RV)

B = biomasse du vert = biomasse VV + biomasse RV

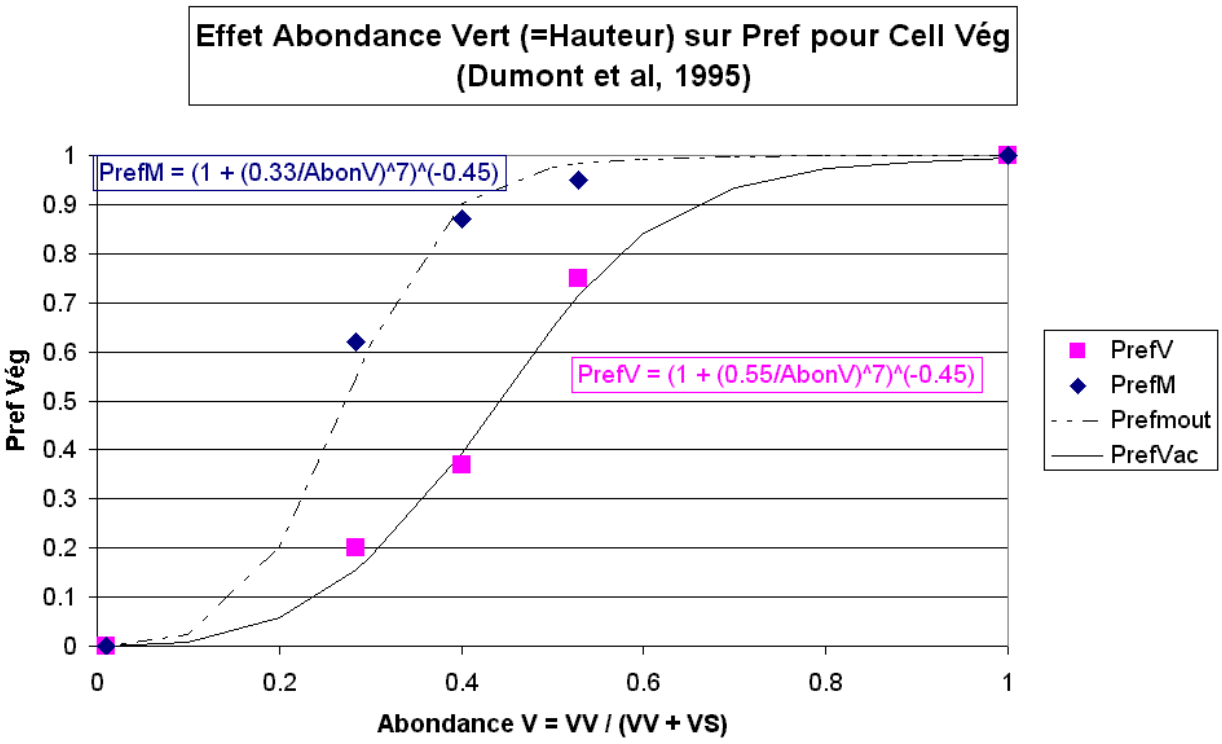
Scénario Empirique 2 **: La valeur de la cellule est une fonction de la préférence des animaux.**

Dumont et al (1995) ont testé les préférences **d'agnelles et de génisses** pour des placettes d'herbe végétative variant par leur hauteur (donc par l'abondance du végétatif vert dans le végétatif total, soit VV/(VV+VS)), par rapport à des placettes d'herbe épiée à différents stades (donc variant par l'abondance du reproductif vert, soit RV/(RV + VV)). A partir de ces tests on peut définir les lois de réponses suivantes :

A) L'animal recherche en priorité les cellules végétatives et évite donc les cellules trop fortement épiées :



B) Pour un stade de végétation donné, l'animal recherche en priorité les cellules qui présentent une forte abondance en végétatif vert :



Une cellule donnée contiendra à la fois du reproducteur et du végétatif, du vert et du sec. L'appréciation de la valeur de la cellule par l'animal sera donc une fonction de PrefStade et de PrefAbonV.

VALi = f(PrefStadei, PrefAbonVi)

Reste à déterminer comment combiner ces deux critères. Dans un premier temps on testera sur un jeu de données les fonctions les plus simples :

VALi = PrefStadei * PrefAbonVi et VALi = ½ * (PrefStadei + PrefABonVi)

Remarques :

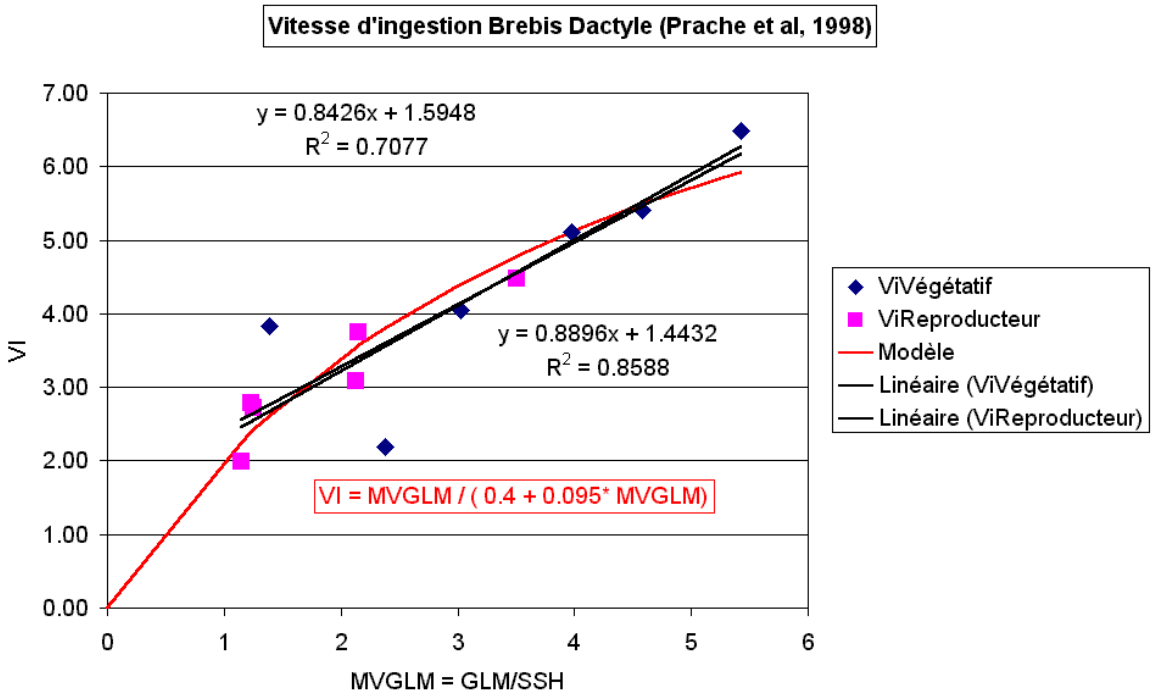
1) Dans le travail de Bertrand, les stades étudiés étaient montaison, épiaison et floraison. Il n'y avait donc pas de RS. On peut essayer d'extrapoler et de considérer : Stade = (RV+RS) / (RV + RS + VV).

2) Le critère AbonV = VV/ (VV+ VS) traduit des différences de hauteur dans le végétatif. C'est comme cela que Bertrand l'a fait varier (7, 11 et 18 cm).

3) Un des intérêts de ce jeu de données est qu'il y a à la fois les ovins et les bovins qui présentent des différences de sélectivité importantes comme le montrent les graphiques.

***Scénario Optimal Foraging Théory* : : L'appréciation de la valeur de la cellule par l'animal est fonction de la vitesse d'ingestion permise par cette cellule.**

Prache et al (1998) ont établi des liaisons pour des couverts végétatifs et des couverts reproducteurs entre la masse de limbes verts (GLM) et la vitesse d'ingestion. Ces liaisons sont distinctes pour les deux types de couverts, à même masse de limbes verts la vitesse d'ingestion étant plus faible sur les couverts reproducteurs. Sophie a fait remarquer que pour avoir des liaisons plus robustes, il faudrait faire intervenir la troisième dimension, soit la hauteur du couvert. En exprimant la vitesse d'ingestion par rapport à la masse de limbes



verts divisée par la hauteur du couvert, soit la masse volumique en limbes verts du couvert (MVGLM), les données obtenues sur les deux types de couverts ne forment qu'une seule population.

On peut donc supposer que la vitesse d'ingestion permise par la cellule dépend de la densité en limbes verts dans le couvert, l'animal (en tout cas la brebis) les recherchant préférentiellement. L'animal va préférer les cellules présentant une densité de limbes verts importante.

Avec MVGLM = Masse Limbes Verts / Hauteur

VAL i = VlPermise i = (MVGLM / Hauteur) / (0.4 + 0.095 * (MVGLM / Hauteur))

VAL i est ensuite recalé par rapport à VAL max. : VAL i = VAL i / VAL max

Remarques :

1) Pour pouvoir utiliser cette relation, il faut pouvoir estimer la masse de limbes vert à partir du compartiment VV qui comprend les feuilles et les gaines. D'après les estimations fournies par P. Carrère :

Masse de Limbes Vert = k * Masse VV Avec **k=0.5** lorsque (RV + RS) = 0 : cellule végétative début de végétation **k=0.85** lorsque (RV + RS) < 25 % de la biomasse : cellule végétative début montaison **k=1** lorsque (RV + RS) > 25 % de la biomasse : cellule reproductrice.

2) Cette relation a été établie pour des ovins. Il faut penser à en établir une pour les bovins. Cela doit être possible à partir des données de Ramon, D'Hour, Petit, Ginane et al.
3) Ce type de relation sera de toutes façon indispensable pour la partie suivante : **quantité prélevée sur la cellule**.

1.2) Distance

Dans le travail de C. Roguet, la distance entre deux stations consécutives réalisées par des brebis ne dépasse pas 60 cm. Il faut définir le nombre de rangées que l'animal a à considérer pour faire son choix et appliquer une fonction qui fait choisir en priorité les cellules

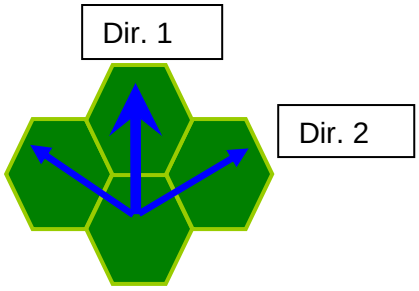
les plus proches. Si on reste sur des cellules de 0.1 m² soit 0.33*0.33, cela fait 2 ou 3 rangées de cellules à considérer.

En se basant sur le principe que l'animal considère les cellules placées devant lui, avec des formes octogonale il a le choix entre 3 cellules (1 rangée de cellules), 8 cellules (2 rangées) ou 15 cellules (3 rangées).

1.3) Direction

L'animal va choisir plutôt les cellules qui se trouvent devant lui que derrière. Comment le modéliser ? On restreint donc le choix des champs des possibles aux 180° devant l'animal. Quelle fonction utiliser pour créer un « indice de direction» ? On pourra partir d'une probabilité de déplacement (cf. MODHETE), avec une probabilité Dir. 1 de partir en face, et une probabilité Dir. 2 de partir à droite ou à gauche. On propose que Dir. 1 soit très grand par rapport à Dir. 2 (0,8, 0,1, 0,1).

Schéma 1 : identification des probabilités de déplacement selon les vecteurs de direction (Dir1, Dir2).



Au final chaque cellule i doit être caractérisée par un indice d'intérêt pour l'animal. Dans le cas le plus simple pour l'instant on a :

$$INTi = VALi * DISTi * DIRi$$

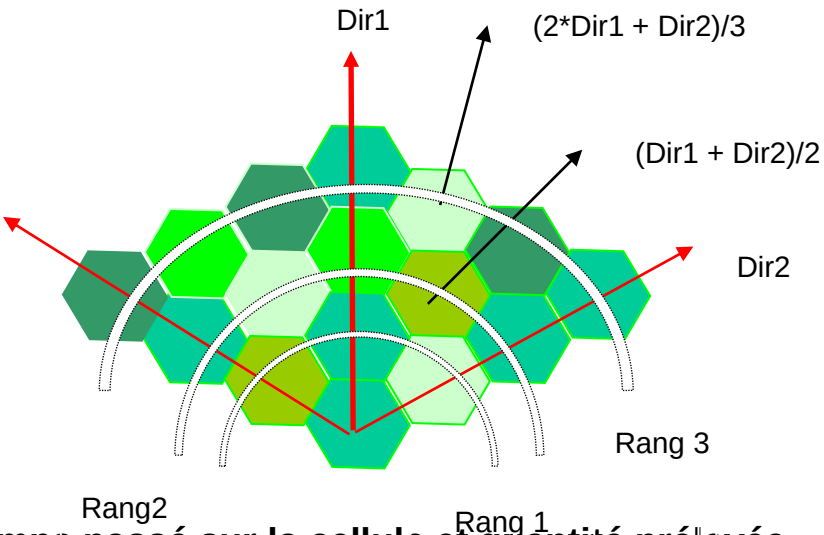
Le choix de l'animal est-il entièrement déterministe, il choisit INTmax, ou bien introduit-on une part d'aléa à ce niveau ? La fonction de choix de l'animal pourra suivre différents scénarios :

- **Matching** : la probabilité de choisir la cellule i est calculée en fonction de la somme des intérêt des n cellules : PCell i = (INTi / Σ INT)

- **Optimal** : on choisit toujours la cellule présentant le plus grand intérêt : Choix cell = max(Inti)

- **Intermédiaire : « over-matching » mais « sub-optimal »**. l'animal a tendance à sélectionner plus que l'intérêt relatif de la cellule (over-matching) mais ne sélectionne pas systématiquement la meilleure (sub-optimal). En pratique cela revient à donner une probabilité de choix a pour la meilleure cellule, b pour la seconde meilleure et c pour la troisième, avec a>>b>>c (ex 0.75, 0.20, 0.05)

Figure 2 : Schémas de détermination de INT i avec identification des trois facteurs : valeur alimentaire de la cellule (VALi, un vert foncé dénote une plus forte valeur alimentaire), la distance au point actuel (DISTi, Rang 1 à 3), la direction (DIRi, combinaison Dir1#Dir2)



2/ Temps passé sur la cellule et quantité prélevée

2.1) Temps passé sur la cellule

Le temps passé et le nombre de bouchées prélevées sur chaque cellule (station alimentaire) est faible (6 à 13 secondes et 6 à 10 bouchées d'après Roguet et al pour des brebis, 7 bouchées pour des bovins d'après WallisDeVries). Les lois d'abandon d'une station alimentaire ne paraissent pas suffisamment connues pour être modélisées de façon

mécaniste. On fait donc le choix d'utiliser une relation statistique pour déterminer le temps passé sur la cellule :

Pour les ovins, on utilisera dans un premier temps les données de C. Roguet et S. Prache.

TempsCell (s) = 6.8 + 4.3 * MasseLimbesVert (T de MS/ha)

Pour les bovins la relation sera paramétrée à partir de la bibliographie (cf. travaux Laca, Distel, Wallis de Vries). La question qui demeure est le choix de la fonction. Devons nous nous contenter d'une relation linéaire (domaine de validité) ou ne doit-on pas rechercher une relation plus générale (hyperbolique).

2.2) Quantité prélevée

De même la quantité prélevée sur la cellule peut être prévue à partir de relation statistiques :

QuantitéIngéréeCell = TempsCell * VICell

VICell = VIPermiseCell / Fonction de Satiété

En effet les vitesses d'ingestion mesurées expérimentalement le sont sur des animaux mis à jeun quelques heures, elles correspondent donc à des vitesses maxi. Comme dans les modèles précédant (Sauvant et al, 1996 ; Cohen-Salmon et al, 1999) on divise cette vitesse par la fonction de satiété de l'animal, dont la valeur varie au cours de la journée avec la réplétion du rumen et le bilan énergétique instantané.

On prendra pour les VIPermises les relations ont été proposées par Prache et al :

VIPermise = MasseLimbesVert / (24.9 + 0.153 MasseLimbesVert) pour cellule végét.
= MasseLimbesVert / (46.8 + 0.202 MasseLimbesVert) pour cellule repro.

Avec **cellule végétative** : compartiments (RV +RS) < 25 % de la biomasse
Avec **cellule reproductrice** : compartiments (RV + RS) > 25 % de la biomasse

Pour les bovins, les équations de temps passé sur la cellule et de vitesse d'ingestion permise restent à établir.

Remarques : i) L'utilisation de ces équations nécessite de distinguer des cellules végétatives et des cellules reproductrices. Il y a donc un seuil à gérer. Dans le travail de

Prache et al, le matériel reproducteur non mort représente au moins 25% de la biomasse totale dans le couvert reproducteur.

ii) L'utilisation de ces équations pose la question de leur robustesse sur d'autres situations que celle qui a servi à les établir. Pour les rendre plus robustes, on peut chercher à introduire d'autres critères dans ces relations (hauteur, densité, à discuter plus avant).

3/ Composition du prélèvement : impact sur la cellule

Le modèle doit prévoir la composition du prélèvement effectué par l'animal sur la cellule, à la fois pour en quantifier l'impact sur les 4 compartiments de la cellule et pour quantifier la valeur de ce qui arrive dans l'animal. Il s'agit donc de prévoir la **sélectivité de l'animal**. A l'évidence il faudra distinguer deux cas de figures ovins et bovins.

Dans un premier temps nous avons retenu les principes suivants :

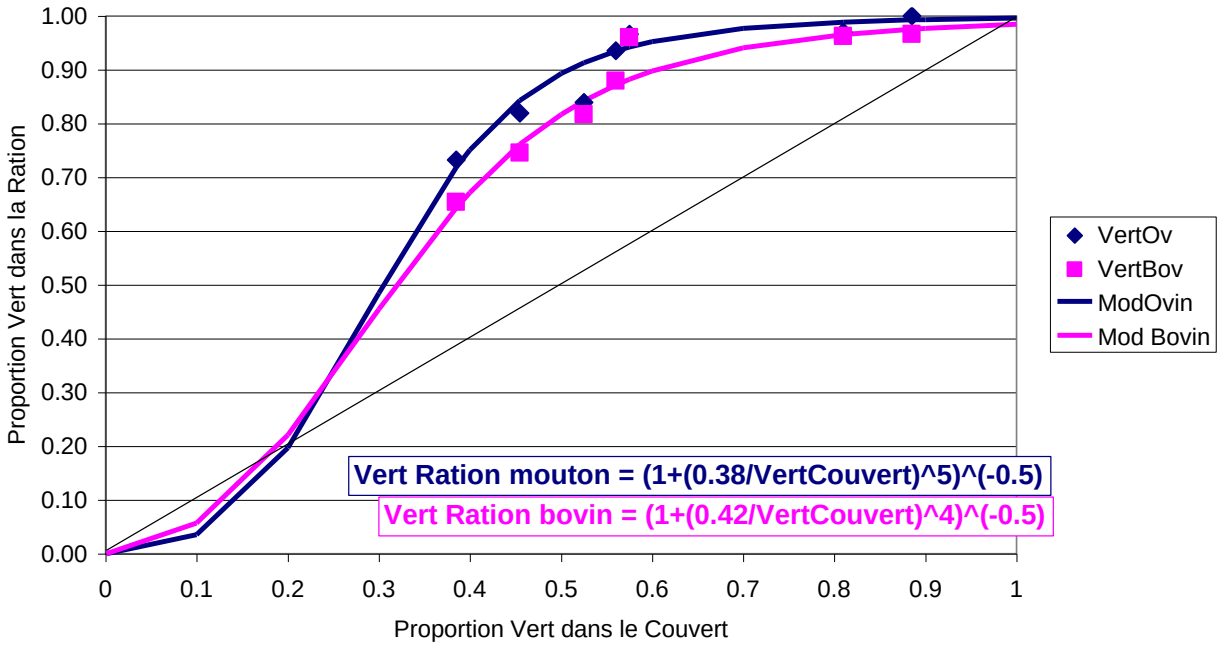
i) L'animal trie en faveur de VV

ii) L'animal ne trie pas par rapport à RV, il le mange en proportion de sa présence dans le couvert

iii) L'animal trie contre VS et RS.

Les données de Jaimieson et Hodgson (1979) permettent de caler dans un premier temps la fonction de tri en faveur de VV pour des ovins et des bovins. Le % de vert dans la ration a été estimé à partir de la dMO des bols oesophagiens des animaux et des dMO du vert et du sec dans le couvert. Les animaux pâturaient des couverts dont le rapport vert/sec variait sensiblement.

D'après Jaimieson and Hodgson, 1979



Rq. : L'utilisation d'une fonction présentant un point d'inflexion permet de traduire le fait que lorsque l'élément recherché devient trop rare, l'animal ne peut plus le sélectionner (cf. Prache, Roguet, Louault et Petit, 1997).

Le calcul de la composition du prélèvement se fait donc ainsi :

Soit **a, b, c et d** les proportions respectives de VV, VS, RV et RS dans le **couvert** et **A, B, C et D** les proportions respectives de VV, VS, RV et RS dans le **prélèvement** de l'animal :

On doit avoir a + b + c + d = 1 et A + B + C + D = 1 donc

A = fonction (a) selon le équations proposées sur la figure

C = c car pas de tri pour ou contre RV *

B = [1 – (A + C)] * b/(b+d)

D = [1 – (A + C)] * d/(b+d)

car l'animal tri contre VS et RS, en proportion de leur présence dans le couvert.

*** Rq. : il se peut que A + c > 1. Dans ce cas, on prendra C = 1 – A et B = D = 0.**

A partir de la quantité ingérée sur la cellule et des proportions des 4 compartiments dans le prélèvement, on peut donc calculer le prélèvement quantitatif réalisé dans chacun des compartiments.

4/ Valeur nutritive du prélèvement

Il reste à quantifier la valeur nutritive du prélèvement réalisé par l'animal. Le modèle animal la caractérise par deux paramètres : Teneur en Parois **(NDF)** et Digestibilité des Parois **(DNDF)**. Il s'agit donc d'estimer ces deux paramètres pour les quatre compartiments ainsi que leur évolution au cours du temps.

Pour faire ces estimations, on s'est basé sur les faciès proposés par Pascal Carrère à partir des données du groupe Agronomie-SAD de Toulouse, sur les tables de la valeur alimentaire (INRA, 1988) et sur des données récemment obtenues par Dulphy et al permettant de faire le lien entre la dMO (digestibilité de la matière organique) et les critères NDF et DNDF.

On a retenu pour l'instant les principes simplificateurs suivant :

1. **La teneur en parois et la digestibilité du compartiment VV est constante au cours du cycle de végétation**. En effet, c'est ce que montrent les données du groupe de Toulouse. Si la digestibilité d'une feuille diminue avec son âge (cf. Duru et al, 1999) la digestibilité du compartiment VV reste pratiquement constante car il y a un constant renouvellement des feuilles. NDF et DNDF du compartiment VV dépendront donc du faciès auquel les cellules appartiennent et qui peut être plus ou moins riche.
2. **De même la teneurs en parois et la digestibilité des compartiments VS et RS sont constantes dans le temps. De plus on affectera les même valeurs à ces deux compartiments**. Les tiges sèches sont assimilables à de la paille, et dans le

travail de Jaimieson et Hodgson on peut estimer la dMO des feuilles sèches à 38% soit une valeur proche de celle de la paille.

3. **Les variations de digestibilité de la cellule au cours du cycle de végétation sont liées au développement progressif du compartiment RV, absent en début de cycle et qui se développe à partir de la montaison jusqu'à la floraison**. Au cours de ce développement la dMO de la plante diminue en moyenne de 0.5 unité par jour (cf. Tables INRA).

A partir de ces trois hypothèse, les simulations qui peuvent être faites à partir des faciès proposés par Pascal montrent une diminution de digestibilité du compartiment RV de environ 0.6-0.7 unité par jour, ce qui correspond à une augmentation d'environ 6-7 g de NDF par kg de MS.

En résumé on peut proposer de commencer avec le jeu de données suivantes :

Faciès riche : dominante Dactyle

VV : NDF = 480 g/kg MS DNDF=0.79
VS et RS : NDF = 850 g/kg MS DNDF=0.40

RV : Si RV < 20 % biomasse alors $NDF_{RV} = NDF_{VV}$ et $DNDF_{RV} = DNDF_{VV}$

Si RV > 20 % biomasse et RS = 0 alors :

$$(NDF_{RV})_{jour\ j} = (NDF_{RV})_{jour\ j-1} + 6\ g/kg\ MS$$

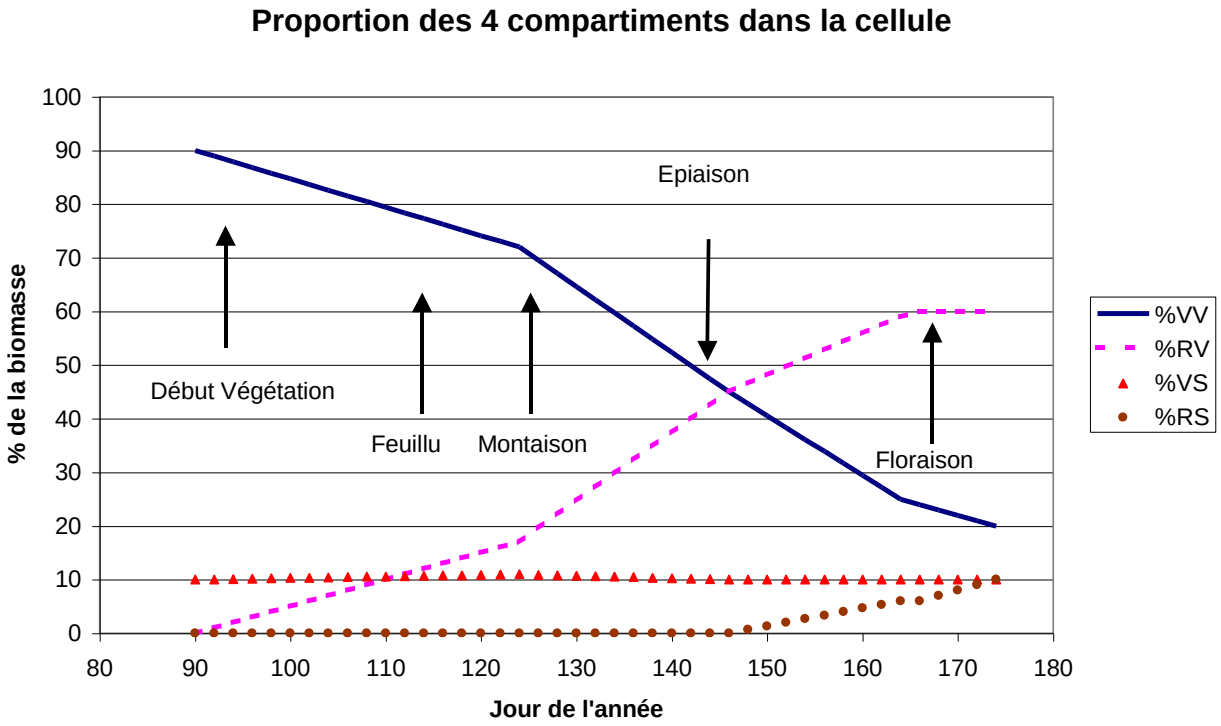
$$(DNDF_{RV})_{jour\ j} = (1/100)*[129.48 - 0.1054 * (NDF_{RV})_{jour\ j}]$$

Si RS >10% alors :

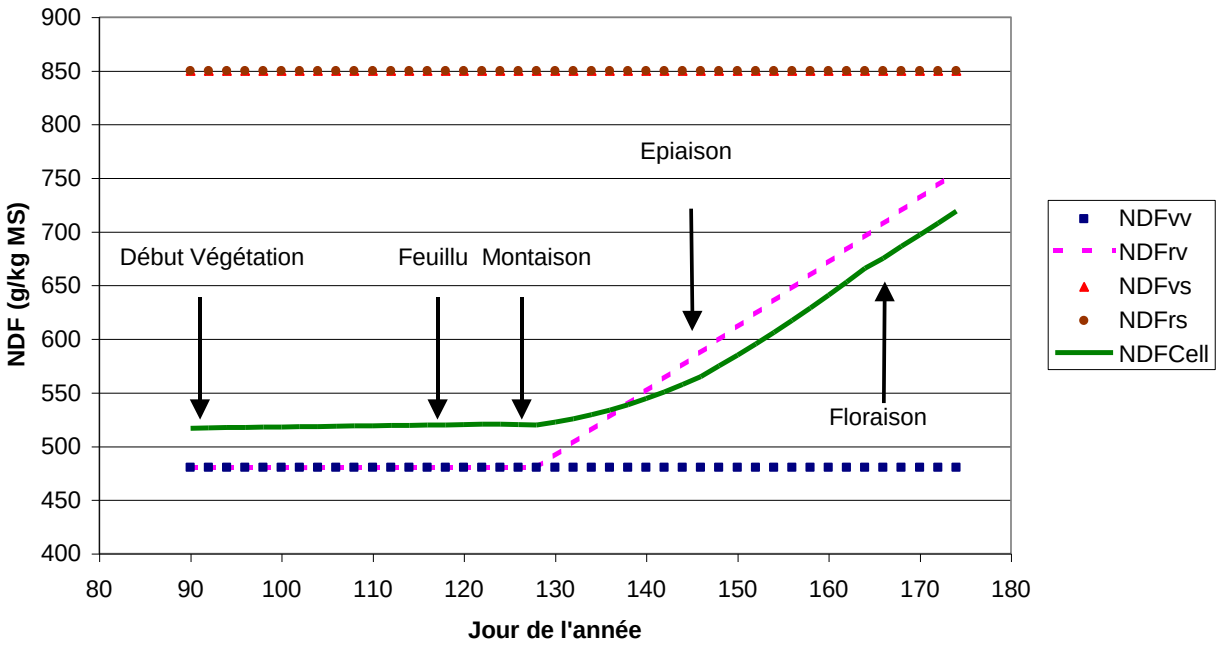
$$(NDF_{RV})_{jour\ j} = (NDF_{RV})_{jour\ j-1}$$
$$(DNDF_{RV})_{jour\ j} = (DNDF_{RV})_{jour\ j-1}$$

Rq. : les valeurs de RV seront limitées par les valeurs plancher de VS et de RS.

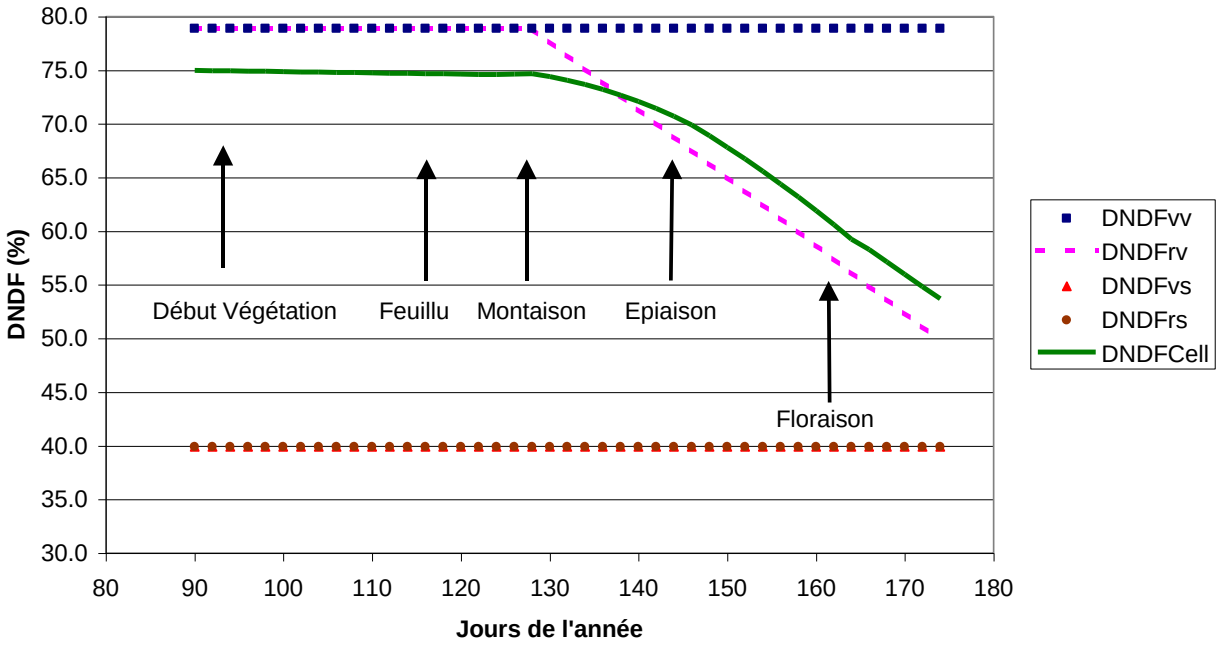
Les graphiques qui suivent montrent pour une cellule d'un faciès à dominante de dactyle (exemple proposé par Pascal) l'évolution de la composition de la cellule, de la teneur en NDF des 4 compartiments et de leur digestibilité. On pourra donc ainsi calculer les teneurs en NDF et les DNDF des prélèvements effectués par les animaux sur les cellules.



Teneur en NDF des 4 compartiments et de la cellule



Digestibilité du NDF des 4 compartiments et de la cellule



Remarques :

1) Ce modèle simple permet de simuler de façon satisfaisante la digestibilité d'une cellule en se basant sur les tables INRA qui donnent la digestibilité de la plante entière.

2) Les valeurs des seuils devront être ajustées pour les différents faciès de végétation.

3) Cette approche nécessite de recalculer chaque jour à minuit la valeur NDF et DNDF des compartiments de la cellule.

Annexe 3 : Les événements du simulateur et leur rôle