Contribution de l'ingénierie dirigée par les modèles à la conception de modèles grande culture

Guillaume Barbier

N° d'ordre : D. U. : 2357

EDSPIC : 611

Université Blaise Pascal – Clermont II

École Doctorale
Sciences pour l'Ingénieur de Clermont-Ferrand

Thèse

Présentée par

Guillaume Barbier

pour obtenir le grade de

Docteur d'Université

Spécialité : Informatique

Contribution de l'Ingénierie Dirigée par les Modèles à la Conception de Modèles Grande Culture

sous la direction universitaire de

Pr. David R.C. Hill
Dr. François Pinet

et pour l'entreprise ITK

Dr. Véronique Cucchi

Soutenue publiquement le 13/06/2013 devant le jury :

Rapporteurs :	Jean Bézivin, Professeur Emérite, Université de Nantes
	Jean-Pierre Müller, Senior Researcher, Cirad, Montpellier
Examinateurs :	Pascal Degenne, Docteur, Cirad, Montpellier
	Philippe Stoop, Dir. Recherche Innovation, ITK, Montpellier
Directeur :	David R.C. Hill, Professeur, LIMOS, Clermont-Ferrand
Co-Directeur :	François Pinet, Dir. de Recherche, Irstea, Clermont-Ferrand

Remerciements

Enfin, après trois années d'efforts, il est temps de mettre un point final à cet ouvrage. J'espère que le lecteur y trouvera quelque élément d'intérêt quant à sa démarche de recherche et/ou d'apprentissage. Les disciplines abordées dans ce travail sont vastes et, à n'en pas douter, insuffisamment couvertes pour leurs spécialistes respectifs, qu'ils m'en excusent.

Je tiens, comme il se doit, à remercier les membres du jury qui m'ont accordé d'atteindre le Graal. En tout premier lieu, les rapporteurs de cette thèse : Jean Bézivin et Jean-Pierre Müller, j'ai apprécié de pouvoir consulter leurs rapports et de pouvoir échanger avec eux lors de la traditionnelle session de questions-réponses. Les échanges auxquels elle a amenés se sont avérés tout aussi stimulants qu'éprouvants.

Ensuite, Pascal Degenne qui a toujours su apporter son sourire et sa pertinence lorsqu'il s'est agi de discuter de ce travail de thèse. La disponibilité dont tu as pu faire preuve, Pascal, a été très appréciée. J'espère pouvoir de nouveau en tirer profit ne serait ce qu'en t'amenant à me présenter Ocelet qui me semble particulièrement intéressant.

Je remercie également Philippe Stoop, Directeur Recherche et Innovation, et par extension la société ITK, pour la confiance qui m'a été accordée dans le cadre de la réalisation de ce travail. J'espère qu'ITK saura y trouver un intérêt.

Je suis bien obligé de faire un cas particulier pour Véronique Cucchi qui a assuré mon encadrement (je sais tu préfères coaching ou peut-être soutien psychologique) au sein de l'entreprise. Tu as repris le bébé au moment où il aurait pu entrer en déshérence. Merci pour… A peu près tout en fait : l'entrain dont tu as su faire preuve, l'écoute, le soutien à tous les niveaux, les repas à la maison, j'en oublie forcément mais l'idée est là. Il est temps que je

te laisse (enfin) respirer pour que tu puisses t'occuper de Lino, le p'tit nouveau, encore une fois toutes mes félicitations et plein de bonheur à vous.

Poursuivons avec mes co-directeurs de thèse : David et François. François, je te dois que ce projet de thèse se soit fait sous ces conditions, certes Clermont ce n'était pas la porte à côté et il m'est arrivé de pester le matin en partant pour plusieurs heures de train, je n'ai pour autant aucun regret, cela m'a permis de rencontrer des personnes passionnantes et de m'enrichir de ces rencontres. Je n'oublie pas non plus ta contribution aux publications qui ont pu se faire jour au cours de ces trois années (merci encore), ni ce tableau blanc de votre salle de réunion indélébilement marqué d'une partie de mon métamodèle (pardon encore).

David, non Benny c'est mieux. Benny donc, que dire ? Je ne sais pas exactement par quoi commencer. L'humanité dont tu fais preuve ? Ton engagement auprès de tes étudiants ? Ta gentillesse ou encore ta disponibilité ? Il y a tant à dire et sans doute pas assez de mots pour t'exprimer ma reconnaissance et mon estime. Cela a été un très grand plaisir de pouvoir vivre cette aventure avec toi comme encadrant. Si je l'ai menée à terme avec soulagement (et beaucoup de ton aide), c'est aussi avec regret que je la vois s'achever parce que je sais bien que l'on aura, à l'avenir, moins de contacts (je ne peux quand même pas t'appeler tous les jours) et qu'il sera sans doute plus difficile pour moi de prendre possession de ton bureau. Je sais bien que je serai le bienvenu (d'ailleurs, tu remercieras encore Anne de ma part pour sa gentillesse et son accueil), tout comme vous le serez, peu importe où l'avenir me portera. Tu feras part de mes amitiés à ton père et de mes remerciements renouvelés pour la qualité de sa relecture sur le book chapter. Merci Benny, merci pour tout.

Et maintenant à qui le tour (sachant qu'il y aura forcément des oublis) ? Sans doute aux autres doctorants (enfin presque tous docteurs maintenant) du labo : Jo P., Romain, Luc, Faouzi, Jo C… Merci pour les accueils, les discussions, les pauses café, les pauses binouzes, l'heure d'attente au resto (non, je ne vise personne). Et puis surtout, Jo et Romain, merci de ne pas m'avoir emmené dans ce train qui, définitivement, n'allait pas à Guimaraes…

Merci à mes collègues d'ITK, qu'ils le soient encore ou soient partis vers de nouvelles aventures. Il y a eu tellement de déménagements (compétition bisannuelle et officielle chez ITK) qu'ils ont à peu près tous été mes co-bureaux à un moment ou un autre. C'est bien difficile de tous vous lister, vous m'excuserez de ne pas me lancer dans l'exercice. Un grand merci pour le quotidien et votre présence en force le jour de ma soutenance, ce fut apprécié. Il y a quand même un de mes futurs collègues que je dois remercier spécifiquement, pour le

stage qu'il a réalisé à mes côtés : Jérémy je te dois une fière chandelle, c'était pas facile mais tu as fait du bon boulot (bon après j'attends que tu reviennes pour te montrer quelques bugs).

Et puis après, il y a les copains, certains ont débuté comme collègues et se sont inévitablement retrouvés dans ce créneau. Marc, Souria et Lollia forcément, je ne peux que la remercier de ses sourires angéliques dont il est difficile de se détacher. Quant à vous deux, vous m'avez apporté votre soutien à maintes reprises. Merci pour votre accueil, votre aide, les bons repas et les soirées jeux et du coup, forcément, merci aux camarades de jeu associés, Céline, Jules, Amandine, Vianney (oui aussi par là (tiens ! Je te mets une double parenthèse pour te faire plaisir (au passage désolé de ne pas faire aussi bien que toi sur les remerciements (bon, quadruple donc. Y'en a un peu plus je vous le mets quand même ?)))).

Thomas, Saliha et Malik c'est un peu du même tonneau, les jeux en moins, les films en plus, je vous garde Malik dès que vous voulez ☺. Anaïs (et puis Simon aussi) merci pour les cours de chant, les discussions (agrémentées d'un verre), les moments où t'es capable de me dire que je suis pénible et j'ose à peine dire les séances de rollers (faudrait qu'on fasse un effort quand même). Lolo et Sylvain, pis leurs petits bouts Jojo et Chloé (je sais bien on va dire que je me répète mais les enfants ca m'éclate, ben eux c'est pareil…) Lolo et Sylvain donc quelque part c'est un peu grâce à moi que vous vous connaissez, je crois que vous me l'avez très largement rendu, votre aide et votre présence ont été inestimables. Il ne faudrait pas que j'oublie les amis du Lisah : David et Tchotcho (longue vie à vous deux) merci pour l'assistance sur le pot de thèse, Michael et tous les autres… Et puis une mention spéciale pour Marc B. et les soirées à refaire l'histoire à couleur ou ailleurs.

Il y a les copines de l'atelier aussi Isa, Aissatou, Marysa, Jéhanne, bon l'indéfectible Pierre aussi même si ce n'est pas une copine. Ma bulle d'oxygène de la semaine, une découverte totale d'un penchant artistique de ma part, mais surtout que de bons moments partagés avec vous, parfois dans le silence religieux des sculpteurs afférés mais bien plus souvent dans une atmosphère de franche rigolade. Merci !

On va garder les derniers pour la fin, forcément ce sont les nouilles, qui, étrangement, ont tendance à s'appeler « poulet-te » entre elles. Des Alex (c'est quand que tu débarques à Montpellier ?), Guigui (tu penseras à dire aux parents qu'on n'a toujours pas nos iPad ?), Poki, Loic, Pabs, Sou, Chouchou, Jeannot (oui je sais t'était pas là cette année mais c'est tout comme), des Matt (toi t'y seras l'an prochain on a déjà voté), Benj, G.O. (oui je sais tous des noms bizarres, moi c'est pareil c'est Juste, heu non, Gino), bref les potes de vacances parfois

mais surtout les potes tout court. Ca commence à faire un moment qu'on se connait mais j'ai l'impression qu'on ne se voit toujours pas assez (oui je sais…). Pis faudrait pas que j'oublie vos marmots quand même, présents ou à venir bientôt… Avec en spéciale dédicace : « JE VEUUUX MON VELO !!! ».

Dans la liste précitée, il y a une Fof qui se cache, je voulais un peu te garder pour la fin, pour tout ce que je t'ai déjà dit et que je redirai encore. Relecture de thèse, soutien psychologique avant la soutenance (honnêtement je pense que seule toi pouvais le faire efficacement), c'est le spécifique à la thèse mais il y a tout le reste… Tu as toujours été là, aux pires comme aux meilleurs des moments (même quand t'étais à l'autre bout du monde), et je suis convaincu que tous ont été mieux de par ta présence… Sinon, il y a Man of Steel au cinoche et il paraît que Superman a une bouille incroyable, ça te dirait d'aller le voir ?

Enfin, je ne peux qu'avoir une pensée pour ma famille, mes parents : merci beaucoup d'être venus c'était loin d'être gagné d'avance. Papa, la vocation scientifique, je sais bien qu'elle vient de toi et franchement sur cette thèse cela m'a bien amusé. Maman, mon petit miracle (je sais je ne t'ai jamais appelée comme ça, mais cela va faire maintenant quelques années que je le pense), merci d'être aussi forte et excuse-moi d'être bête parfois. Merci aussi à mes sœurs et mes beaufs (bon y en a un qui a pas encore le titre officiel mais ca arrive bientôt paraît-il), mes neveux Théo et Victor, Gégé, Papi et Mamie, tous ceux que je ne peux que voir lorsque je rentre « à la maison ». J'ai beau être loin, je n'oublie pas mes attaches et même si je ne suis pas très bon pour l'exprimer, vous êtes mon foyer, là où il fait toujours chaud. Je n'oublie pas non plus ceux qui sont partis et qui manquent…

Sommaire

Table des illustrations

Introduction

La discipline de la modélisation et de la simulation numérique est relativement jeune quel que soit son domaine d'application. En sciences du vivant, des relations mathématiques étaient déjà établies en phyllotaxie à la fin du XVIII^{e} siècle (Varenne 2010b). Cependant, ces approches n'étaient que descriptives sans notion de causalité et sans explication du phénomène observé. A notre connaissance, ce sont les travaux d'Alan Turing (Turing 1952) qui ont abouti au premier modèle mathématique du vivant exploré par simulation à l'aide d'un ordinateur. Cette notion d'exploration se réfère à la découverte. En effet, bien que les représentations mathématiques soient définies par l'être humain, ce sont les capacités d'intégration numérique de l'ordinateur qui vont permettre d'obtenir une résultante du modèle qui s'avère inaccessible à l'esprit humain. Avec l'avènement de l'ère informatique, les approches par modélisation et simulation se sont développées y compris dans la représentation de la croissance et du développement des cultures en agronomie.

L'activité de modélisation des cultures a suivi la même trame que le développement des pratiques et des technologies en ingénierie logiciel. Tout d'abord, des langages procéduraux tels que le Fortran ont été utilisés pour l'implémentation des modèles (Bouman *et al.* 1996). L'apparition de la programmation orientée objet, grâce aux recherches en simulation dès 1962 avec Simula I, puis Simula 67 (Dahl et Nygaard 1966), a conduit à l'adoption de ce paradigme et à une recherche active sur la thématique de la généricité (Gauthier *et al.* 1999). En parallèle, différentes approches se sont orientées vers la modularité telles que pour les outils APSIM (McCown *et al.* 1996) et DSSAT (Jones *et al.* 2001). L'intérêt de la modularité est encore défendu dans les conceptions les plus récentes de plateformes de modélisation et simulation pour la croissance des plantes (Fournier *et al.* 2010). En suivant cette trame, la

prochaine étape technologique devrait être l'adoption progressive de ce qui représente l'avenir de l'ingénierie logiciel : l'Ingénierie Dirigée par les Modèles (IDM), introduite au début des années 2000.

D'autres domaines de la modélisation et de la simulation ont déjà recours à l'IDM (deLara et Vangheluwe 2002a; Touraille *et al.* 2011). Dans certains cas, ces démarches ont été mises en place par des initiatives conjointes du monde de la recherche académique et de celui de l'industrie comme ce fut le cas pour Modelica pour la représentation de systèmes mécaniques et électroniques (Mattson et Elmqvist 1997). Dans le cas de Modelica, un double constat a mené à son développement :

- l'inadéquation des bonnes pratiques de la programmation orientée objet dans le cadre de modèles dirigés par des équations ;
- la nécessité de réaliser un prototype du modèle scientifique avant d'en obtenir une implémentation utilisable en production.

Dans le domaine de la modélisation des cultures, les problématiques de type industriel sont plus rares. En effet, les démarches de modélisation de l'ordre de la recherche académique tendent à ne considérer qu'une seule implémentation du modèle. Le nombre d'acteurs industriels intéressés par les aspects de modélisation des cultures est relativement faible, la plupart n'utilise les modèles de culture qu'en tant qu'outils internes de recherche et développement, ces modèles gardant alors le statut de prototype. Pour l'entreprise ITK, la situation est toute autre.

ITK propose des solutions innovantes d'aide à la décision pour les exploitants agricoles, techniciens et consultants. Ces outils d'aide à la décision, orientés web, exploitent des modèles mécanistes. Ces modèles prédisent l'évolution du statut d'une culture en fonction des conditions agro-environnementales rencontrées au cours de la saison et emploient un système expert pour l'aide à la décision.

Dans ce contexte, ITK fait face à plusieurs défis concernant la production de modèles. Tout d'abord, la conception de prototypes de modèles scientifiques est réalisée par un groupe d'experts dans un environnement Matlab®. Ces prototypes sont ensuite traduits en code Java par des ingénieurs logiciel afin de pouvoir intégrer les modèles aux systèmes experts. S'agissant de la représentation de systèmes vivants à une échelle annuelle, les temps de conception, de validation et de test *in situ* des modèles sont relativement longs. Cela oblige à

maintenir et à faire évoluer en parallèle le prototype du modèle ainsi que son implémentation Java. Tant les opérations de traduction des prototypes vers la plateforme Java que la maintenance en parallèle des deux implémentations sont à l'origine d'une perte de productivité. De plus, le risque d'apparition de disjonctions entre les deux implémentations ne peut être négligé. Cela est d'autant plus vrai que l'absence d'outils formels dédiés à la modélisation conceptuelle, telle que soulevée par Robinson (Robinson 2007), conduit à ce que le code du prototype devienne la référence à l'exclusion de toute documentation.

Dans le cadre de notre travail de thèse, nous nous sommes saisis de cette problématique de production et avons souhaité y apporter une solution par l'emploi des techniques issues de l'IDM. L'objectif, que nous nous sommes fixés, est d'obtenir à terme un environnement de modélisation et simulation dédié à la croissance des plantes et permettant d'obtenir une implémentation Java des modèles sans intervention d'ingénieurs logiciel.

Afin de bien les faire percevoir à notre lecteur, le premier chapitre dévoile tout d'abord les éléments de contexte qui ont déterminé notre problématique et l'approche retenue. Après une brève discussion de la notion de modèles, nous donnons les éléments nécessaires à la compréhension de la modélisation grande culture, qui est une des catégories de modèles de croissance de plantes. Cette catégorie correspond aux modèles employés par ITK dans ses outils d'aide à la décision. Nous présentons donc les caractéristiques du processus de production de ces modèles dans l'entreprise et la problématique qui en découle.

Au chapitre 2, nous décrivons les notions et pratiques associées à l'ingénierie dirigée par les modèles et montrons en quoi elles peuvent répondre à la problématique définie grâce à la mise en place d'un DSML (Domain-Specific Modelling Language) et de ses éditeurs associés. Ensuite, compte tenu de notre problématique, nous dressons un inventaire non exhaustif de différentes solutions de modélisation et simulation, dédiée ou non à la croissance des plantes, à caractère commercial ou issue de la recherche académique, susceptibles de répondre à nos attentes. Nous montrons alors les points forts et les faiblesses de ces différentes solutions quant aux objectifs fixés. Nous justifions ainsi leur inadéquation et la mise en place d'un DSML dédié au domaine de la M&S grande culture.

Le chapitre 3 expose la démarche d'ingénierie dirigée par les modèles que nous avons mise en œuvre. Après une identification des caractéristiques d'intérêt de la modélisation grande culture, nous proposons une formalisation des concepts qui en découlent sous la forme d'un métamodèle. Ces concepts ont également été intégrés dans un cadriciel en langage Java dédié

à la modélisation grande culture dont nous donnons les caractéristiques majeures. Enfin, nous explicitons la syntaxe graphique retenue pour notre DSML ainsi que les éditeurs nécessaires à l'obtention d'un prototype pour la conception de modèles grande culture.

Finalement, c'est au cours du chapitre 4 que sont présentées les étapes techniques qui ont conduit à l'obtention de ce prototype et de la génération de code Java. Un cas d'utilisation de notre prototype à partir d'un modèle de croissance de la vigne utilisé par ITK est détaillé. A partir de ces réalisations, nous discutons le travail effectué eu égard à la problématique définie et livrons les perspectives envisagées.

Chapitre 1 - Eléments de contexte

Ce travail de thèse s'inscrit à la croisée des chemins entre différentes disciplines, l'informatique et l'agronomie, mais également entre différents domaines : l'ingénierie dirigée par les modèles, la physiologie végétale, la modélisation en agronomie et dans une acception plus large la modélisation et la simulation (M&S) numérique. Certaines terminologies faisant appel à des notions d'agronomie ou de physiologie végétale peuvent être méconnues du lecteur, dès lors nous nous efforcerons au cours de cette partie de faire apparaître ces termes en caractères gras et d'en fournir une définition en bas de page lorsque nécessaire. Chacun de ces termes et sa définition sont intégrés dans le glossaire fourni en annexe.

1- Discussion autour de la notion de modèle

Il apparaît délicat de présenter des travaux sur l'utilisation de l'ingénierie dirigée par les modèles (IDM) dans le domaine de la modélisation en agronomie sans aborder en préalable la notion de modèle. Parmi les usages communs, Le Petit Larousse définit un modèle comme « Ce qui est donné pour servir de référence, de type. Modèle d'écriture » et encore « Ce qui est donné ou choisi pour être reproduit. Copier un modèle ». Suivant ces acceptions, le modèle est donc un élément concret, tangible, qui va servir à produire un écrit selon le premier exemple donné ou un duplicata du modèle selon le second.

D'autant plus trompeuse, par rapport au sens scientifique du mot modèle, est la terminologie du monde artistique où le peintre ou le sculpteur utilise un modèle vivant pour produire leur œuvre. En effet, dans ce sens, le modèle est un objet concret dont l'artiste cherche à produire une abstraction. Or, du point de vue scientifique, le modèle correspond à la représentation d'un système, représentation qui devient synonyme d'abstraction. Un véritable antagonisme

existe dans ces deux acceptions du mot modèle. Compte tenu du contexte de rédaction de ce mémoire, nous nous focaliserons sur la dernière : le modèle en tant qu'abstraction.

Une définition, incontournable, du terme modèle a été fournie par Marvin Minsky « *To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A* » (Minsky 1965). De cette définition, plusieurs éléments transparaissent. En premier lieu, un observateur, le modélisateur, s'intéresse à un objet A. A est donc un système sur lequel le modélisateur va porter son intérêt, intérêt qui se traduit par des questions que le modélisateur se pose sur A. Il cherche à répondre à ces questions en utilisant A*. L'exposé de Minsky explique clairement que A* est une construction mentale que réalise le modélisateur. Cette construction mentale sous-entend que le modélisateur effectue une projection du système en ne retenant que les éléments permettant de répondre aux questions qu'il se pose. Il s'agit donc d'une simplification du système considéré. Cette notion de simplification fait défaut dans la définition de Minsky.

Pour illustrer l'importance de la notion de simplification, nous proposons, à l'instar de Jean Bézivin (Bézivin 2005b), de considérer ce passage tiré de Sylvie and Bruno concluded, dans lequel Lewis Caroll narre une conversation sur la réalisation de cartes (et donc de modèles) de plus en plus détaillées :

> "That's another thing we've learned from your Nation," said Mein Herr, "map-making. But we've carried it much further than you. What do you consider the largest map that would be really useful?"
> "About six inches to the mile."
> "Only six inches!" exclaimed Mein Herr. "We very soon got to six yards to the mile. Then we tried a hundred yards to the mile. And then came the grandest idea of all! We actually made a map of the country, on the scale of a mile to the mile!"
> "Have you used it much?" I enquired.
> "It has never been spread out, yet," said Mein Herr: "the farmers objected: they said it would cover the whole country, and shut out the sunlight! So we now use the country itself, as its own map, and I assure you it does nearly as well.

De manière humoristique, ce passage nous montre qu'un modèle, sans simplification, présente peu d'intérêt. Nous irions même jusqu'à dire qu'il est utopique.

Le modèle du grand tout n'existe pas, dans son article, Minsky évoque le problème conceptuel de modéliser entièrement le monde, le modèle en faisant partie, ceci signifierait

que le modèle soit à même de s'auto-décrire. Même si le modélisateur se focalise sur un système restreint (une portion du monde), la représentation qu'il en fera ne sera qu'une approximation :

- De par le fait qu'un système fermé n'existe pas, des échanges sont réalisés entre le système et son environnement. En M&S, les éléments extérieurs, tels que la température de l'air dans lequel se situe le système, qui influent sur le système, sont appelés variables de forçage. Une variable de forçage peut être continue dans le temps, cependant sa mesure ne peut être que discrète dans le temps. Dès lors interviennent des hypothèses simplificatrices pour permettre de s'appuyer sur ces données (interpolation, moyenne des valeurs). Le modèle comprend alors une simplification de la relation entre le système et son environnement.
- Et également par le fait qu'un système n'est jamais décrit dans son entièreté. Il est possible de mettre en équations un mécanisme d'horlogerie sans pour autant avoir à prendre en compte la composition atomique de ses rouages. Outre la difficulté, voire l'impossibilité, de la tâche, cela ne présenterait que peu d'intérêt pour modéliser le fonctionnement de ce système mécanique. L'intérêt de la démarche détermine donc les choix de simplification opérés par le modélisateur.

Ce lien, entre les choix de simplification du modélisateur et l'intérêt qu'il porte au système étudié, est plus explicite dans la définition fournie par David R.C. Hill dans (Hill 2000, 2009) : « *Un modèle est une abstraction qui simplifie le système réel étudié en ignorant de nombreuses caractéristiques de celui-ci, pour se focaliser sur les aspects qui intéressent le modélisateur et qui définissent la problématique du modèle* ». La démarche du modélisateur se focalise donc sur certains aspects de son système d'étude. Pour pouvoir définir son modèle, il opère des choix sur les éléments qui lui paraissent pertinents vis-à-vis des objectifs fixés. Ces choix soulèvent deux points.

Le premier est qu'ils sont réalisés dans la limite de la connaissance du système étudié. Ceci peut amener le modélisateur à identifier les manques dans la connaissance du système étudié par l'impossibilité de reproduire par la simulation le comportement connu de ce système. Une autre possibilité est d'introduire un mécanisme hypothétique afin de montrer si ce mécanisme, non identifié empiriquement, permet de reproduire le fonctionnement du système. De telles démarches heuristiques sont de bons moyens d'orienter la recherche sur les systèmes réels et

sont particulièrement utiles dans la compréhension du vivant dont les systèmes complexes restent un défi à la théorie.

Le deuxième point d'importance, quant aux choix réalisés par le modélisateur, est qu'ils impliquent une certaine subjectivité dans la démarche de modélisation. Une bonne reproduction, par le modèle, du comportement du système n'implique pas que le modèle est une fidèle représentation des mécanismes qui se produisent au sein même du système. C'est le sens du propos d'Alan Turing dans (Turing 1952) : « *This model will be a simplification and an idealization, and consequently a falsification* » (p.37). Le modèle ne comporte pas la vérité absolue, il permet de répondre à des questions concernant le système, de tester la validité d'hypothèses ou encore de prédire le fonctionnement du système mais toujours dans la limite du domaine pour lequel il a été testé et validé. Il est donc important pour le modélisateur de conserver un regard critique sur l'outil qu'il a mis en place. La citation de Martin Wick dans (Tukey 1962) reste d'actualité : « *The hallmark of good science is that it uses models and 'theory' but never believes them* » (p.7).

Les exemples donnés jusqu'à présents sont intentionnellement issus du domaine de la modélisation et simulation. L'activité de modélisation en ingénierie logiciel nous semble différer quelque peu de par la nature même du système modélisé. En effet, le système correspond alors à un programme informatique, or un programme informatique n'a pas d'existence tangible, en soi il s'agit d'une abstraction (Kramer 2007). Si le programme est une abstraction, exprimée dans un langage donné, ne peut-il être considéré comme un modèle ? Jean Bézivin, dans un exposé clair et concis (Bézivin 2004), nous permet de comprendre que le système n'est pas le programme cible mais l'exécution de celui-ci, cette subtilité peut échapper à nombre de praticiens de l'ingénierie logiciel. Ce qui peut amener à chercher à définir des différences entre langages de modélisation et langages de programmation, comme dans cet article de Sun et coauteurs (Sun *et al.* 2008), alors que conceptuellement il n'existe que des langages de modélisation. Il pourrait être intéressant de philosopher sur la question suivante : « L'utilisation d'un langage, quel qu'il soit, n'est elle pas œuvre de modélisation ? ». Nous laisserons la question ouverte à la réflexion de chacun.

Quant à la distinction entre le modèle de l'ingénierie logiciel et le modèle de la modélisation et simulation, la seule différence notable, à notre sens, est déterminée par l'intention du modélisateur :

- en ingénierie logiciel : le modèle a comme objectif de définir le fonctionnement du système à créer ou existant (Kühne 2006) ;
- en M&S : le modèle a pour but de décrire un système existant, pour acquérir de la connaissance sur celui-ci ou prédire son évolution, ou encore de prédire le fonctionnement d'un système qui doit être créé (Viswanathan *et al.* 2008; Duchaine *et al.* 2009).

Il serait possible de discuter plus longuement de la notion de modèle, tant les disciplines qu'elle touche sont variées et empreintes de leurs propres méthodes et formalismes. Pour résumer, nous définissons un modèle comme : une représentation simplifiée d'un système, existant ou non, dont le contenu et les modalités d'expression sont guidés par l'intention du modélisateur et, donc, suivant l'objectif qu'il cherche à atteindre.

Comme toute autre définition, elle pourra vraisemblablement être mise en défaut, cependant nous la pensons suffisamment précise pour les deux domaines de modélisation abordés dans ce travail de thèse : celui de l'ingénierie logiciel et celui de l'agronomie. C'est ce dernier domaine que nous allons aborder dans la partie suivante, un bref historique des approches de modélisation sera dressé avant d'aborder plus en détail l'une d'elles : la modélisation à base de processus.

2- La modélisation en agronomie

L'**agronomie** (du grec agros : champ et nomos : loi) correspond à « [l'] étude scientifique des relations entre les plantes cultivées, le milieu (sol, climat) et les techniques agricoles » (Petit Larousse, ed. 2011). Afin de bien comprendre le contexte et les enjeux de la modélisation en agronomie, nous allons tout d'abord présenter quelques éléments de connaissances générales de physiologie végétale. Après un bref historique de la modélisation dans le domaine, nous aborderons plus en détail la modélisation à base de processus.

2.1- Quelques éléments de connaissance générale en physiologie végétale

Les plantes cultivées en agriculture font partie des **végétaux supérieurs (embryophytes)**. Ce sont essentiellement leurs **structures reproductrices** (fleurs : chou-fleur, brocoli ; fruits : pomme, tomate, courge) ou leurs **organes de réserves** (tubercules tels que le radis, la pomme de terre, l'igname) qui sont exploités à des fins d'alimentation, humaine ou animale, la plus grande partie du temps mais aussi pour l'industrie (textile : chanvre, coton ; énergie : biocarburants issus du colza). Les embryophytes sont aussi appelés plantes à tige. Elles présentent un **appareil racinaire** et un **appareil caulinaire**, la tige. Cette décomposition existe déjà dans la graine (*Schéma graine plantule radicelle et premières feuilles*). Chacun de ces deux appareils remplit des fonctions bien précises pour la plante.

2.1.1- L'appareil caulinaire

La tige, ou système aérien, porte les feuilles et, pendant une partie de la saison, les structures reproductrices de la plante. La partie aérienne assure la **photosynthèse** qui consiste en la production de matière organique carbonée à partir de l'énergie lumineuse et du dioxyde de carbone atmosphérique. La photosynthèse s'effectue majoritairement au niveau des feuilles. Les feuilles sont également les organes qui assurent la transpiration, fonction biologique essentielle. En effet, la transpiration, par effet de succion, va assurer la remontée des minéraux et de l'eau absorbés par les racines vers les feuilles où ils seront utilisés pour la photosynthèse.

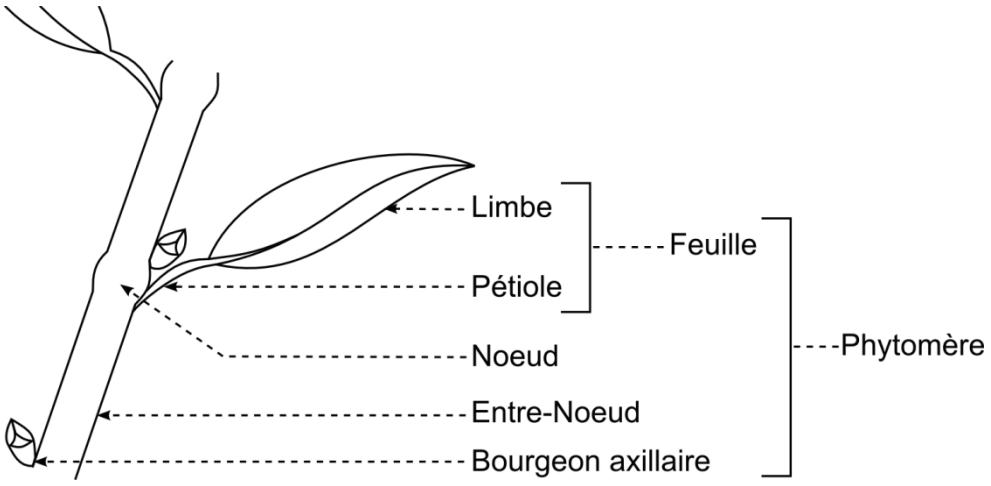


Figure 1.2.1 : Schéma d'un phytomère

Une tige consiste en une répétition d'unités de base (cf. figure 1.2.1) : les **phytomères**. Chaque phytomère se décompose en un nœud (limite avec un autre phytomère), un entre-nœud (partie de l'axe de la tige située entre deux nœuds), une feuille s'insérant au niveau du nœud avec à son aisselle (entre le **pétiole** de la feuille et la tige) un **bourgeon axillaire**.

Le **méristème apical** est responsable de la mise en place de nouveaux organes sur l'axe primaire de la plante. Au début de la saison, il permet de produire de nouveaux phytomères. *In fine* il aboutit à la production d'une structure reproductrice : une fleur ou un ensemble de fleurs. Les bourgeons axillaires peuvent conduire à la mise en place d'un axe secondaire (une tige sur la tige), dont le fonctionnement est, en tous points, identique à celui de la tige principale.

2.1.2- L'appareil racinaire

L'appareil racinaire constitue la surface d'échange de la plante avec le sol, il permet à la plante de trouver eau et éléments minéraux nécessaires à son entretien et à sa croissance. En termes de quantité, il s'agit majoritairement de l'eau et de l'azote. Outre ce rôle dans la nutrition de la plante, le système racinaire assure aussi un rôle d'ancrage dans le sol qui permet à sa partie aérienne de se maintenir.

Cela dit, et comme les choses ne sont jamais simples en biologie, nous tenons à signaler que toute partie d'une plante qui se trouve dans le sol n'est pas forcément une racine, la pomme de terre se récolte dans le sol mais n'en est pas moins une tige. Les yeux de la pomme de terre sont en effet des ébauches de feuilles avec des bourgeons, une pomme de terre laissée au soleil verdira, ceci est preuve de la présence de **chlorophylle** qui n'est pas produite par les racines. De même, les racines d'une orchidée **épiphyte** ne verront-elles jamais le sol puisqu'elles se développent dans certains cas à plusieurs dizaines de mètres au-dessus de celui-ci (ex : *Angraecum sesquipedale*, l'étoile de Madagascar)

2.1.3- La phénologie

Comme nous avons pu le voir précédemment, la mise en place des organes aériens est due à l'activité des bourgeons apicaux qui, *in fine*, produiront des fleurs. En cours de saison, la plante va subir des modifications qui entrainent, entre autres, le changement du fonctionnement des bourgeons, ceux-ci arrêtent de produire des phytomères pour donner des organes floraux. Ces changements saisonniers, tout comme la germination de la graine, le **débourrement** des bourgeons, sont déterminés par des facteurs environnementaux. La

phénologie a pour objet d'étudier ces changements. Pour les espèces cultivées, des échelles phénologiques ont été définies afin de repérer des stades clés qui révèlent des modifications dans le fonctionnement des organes de la plante : stades et changement de fonction des organes correspondent au **développement**[1] de la plante. La figure 1.2.2 nous montre la succession de stades phénologiques pour la vigne. Il faut noter que l'échelle phénologique est souvent spécifique à l'espèce considérée et que différentes échelles, issues de différentes écoles, peuvent exister pour une seule et même espèce.

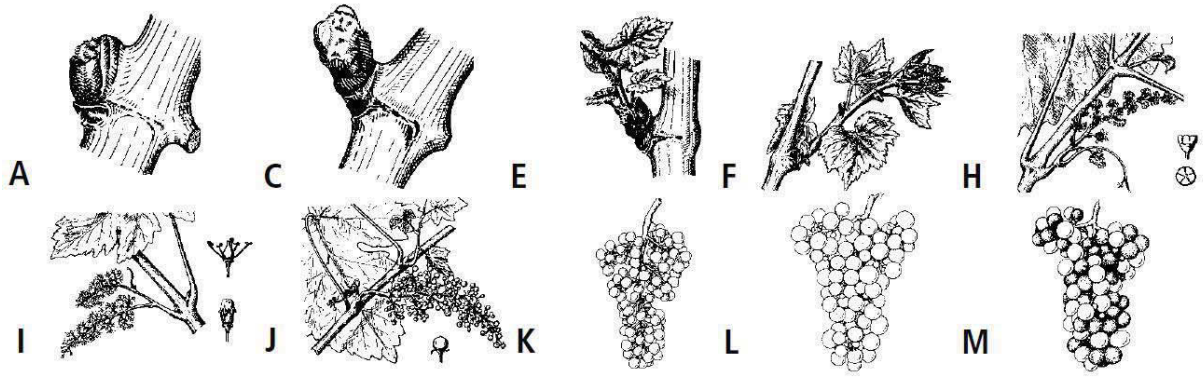


Figure 1.2.2 : Différents stades phénologiques de la vigne suivant l'échelle de Baggiolini (Baggiolini 1952)

Cette partie est loin d'être exhaustive ; cependant, elle offre les éléments clés nécessaires à la bonne compréhension de la suite de ce mémoire. Le lecteur pourra se référer à (Heller *et al.* 2004a) et (Heller *et al.* 2004b) pour en connaître plus sur la physiologie végétale.

2.2- Histoire de la modélisation en agronomie

L'application des mathématiques aux systèmes végétaux n'est pas un phénomène nouveau. Comme le note Frank Varenne dans son excellent ouvrage historique et épistémologique (Varenne 2010b), elle a débuté, entre la fin du XVIII^{e} et le début du XIX^{e} siècle, par une approche arithmétique descriptive de la **phyllotaxie**[2]. Est alors évoqué le terme de « loi mathématique », ces lois mathématiques sans explication ni application ne peuvent être appelées modèles.

Les premières réelles approches de modélisation pour le monde végétal ont été menées dans le domaine statistique. Parmi celles-ci, l'étude de Gregor Mendel (1822-1884) lui ont permis

[1] Le développement d'un être vivant désigne l'ensemble des stades et changements physiologiques associés qu'il rencontre au cours de sa vie.

[2] La phyllotaxie désigne l'étude de la disposition des feuilles et des ramifications des plantes.

d'être considéré, à titre posthume, comme le père fondateur de la génétique. Ses travaux se sont focalisés sur l'**hybridation**[3] de différentes variétés de pois et la reproduction entre individus résultant du premier croisement. Cette étude a mené à la définition des trois lois de Mendel sur l'héritabilité des caractères. Bien que menée par un botaniste et sur des plantes, il ne s'agit pas à proprement parler d'une modélisation dans le domaine de l'agronomie. Cependant il nous semblait utile de rappeler l'importance de la découverte de Mendel. Parmi les statisticiens ayant fortement contribué à la **biométrie**, l'un des principaux est sans aucun doute Ronald Aylmer Fisher. Ses premiers travaux menés à la station expérimentale de Rothamsted ont permis de mettre à profit des années de données expérimentales (Fisher 1921; Fisher et Mackenzie 1923). Plus d'informations sont accessibles dans (Varenne 2010a) sur l'approche de modélisation statistique à laquelle Fisher a œuvré.

Ce n'est qu'après-guerre que la modélisation mathématique a pu voir le jour notamment grâce aux travaux d'Alan M. Turing (Turing 1952). Par une mise en équation de la diffusion de signaux chimiques au niveau d'un amas de cellule, Turing visait à montrer qu'il était possible de décrire la **morphogénèse**[4] des fleurs ainsi que d'êtres vivants tels que le polype d'eau douce. Le premier tableau fournit dans sa publication contient des chiffres obtenus principalement à l'aide d'un ordinateur. C'est, à notre connaissance, le premier cas d'utilisation d'un ordinateur pour la modélisation en sciences de la vie.

C'est l'avènement de l'ère informatique qui a permis le développement de la recherche autour des modèles mathématiques pour l'agronomie. En premier lieu, ces modèles ont été utilisés pour tester les connaissances autour des processus physiologiques se déroulant dans la plante (Loomis *et al.* 1979; Bouman *et al.* 1996; Sinclair et Seligman 2000; Wallach 2006). Ces démarches s'inscrivent dans une représentation mécaniste de systèmes biologiques. Des formalismes mathématiques sont employés pour représenter des processus biophysiques. Ce type de modèle est communément appelé modèle mécaniste ou modèle à base de processus. Le champ d'application originel s'est focalisé sur les cultures céréalières d'où la troisième appellation : modèle grande culture. Dans la suite de ce mémoire nous utiliserons cette dernière pour désigner ces modèles, dans la pratique, l'appellation anglaise *crop model* est également couramment employée.

[3] L'hybridation désigne le croisement (i.e. la reproduction) entre deux espèces ou deux variétés différentes d'une même espèce.

[4] La morphogénèse correspond à l'ensemble des processus permettant à un organe ou un être vivant d'atteindre sa forme finale.

De Wit, avec ses travaux sur la respiration et la photosynthèse (de Wit *et al.* 1970), est généralement considéré comme l'un des pères fondateurs de la modélisation des cultures (van Ittersum *et al.* 2003; Brisson *et al.* 2006). Ses travaux ont donné naissance à « l'école de de Wit », rassemblant essentiellement des chercheurs de l'université de Wageningen. Le développement de ELCROS (ELementary CROp Simulator) (de Wit *et al.* 1970) a servi de base pour le développement de BACROS (Basic CROp growth Simulator). De manière contemporaine, des travaux menés sur le rôle de l'architecture du couvert végétal dans l'interception lumineuse (Baker et Meyer 1966) ont conduit à la mise en place d'un modèle pour le coton : SIMCOT (Duncan 1973). Tout comme les modèles précédents, ARCWHEAT (Weir *et al.* 1984) était destiné à l'évaluation de la **croissance potentielle**[5] d'une culture (ici, le blé). A partir de ces modèles, des améliorations ont été apportées afin de pouvoir prendre en compte les disponibilités en eau et en azote, l'azote est considéré comme l'élément nutritif limitant dans la croissance des plantes. ELCROS a ainsi servi de base à la réalisation de SUCROS (Simple and Universal CROp growth Simulator) et de PAPRAN (Production of Arid Pastures limited by RAinfall and Nitrogen) (Bouman *et al.* 1996). Dans d'autres cas, la réécriture complète du modèle a été nécessaire, il s'agit notamment de l'abandon de SIMCOT pour le développement de GOSSYM (Baker *et al.* 1986). L'amélioration des connaissances sur les systèmes modélisés, la manière de les représenter, l'ajout des facteurs limitant, tels que le stress hydrique ou azoté, ont permis d'envisager l'utilisation des modèles à des fins prédictives. Ces prédictions ont abordé à la fois des conditions de plein champ correspondant aux domaines de validation des modèles ou à l'exploration de conditions non encore testées. Dans le premier cas, le modèle associé à un moteur de décision a donné lieu à la création des premiers outils d'aide à la décision (Bouman *et al.* 1996; Jones *et al.* 2006). Le second cas a une vocation plus exploratoire, il peut permettre l'identification des meilleurs itinéraires techniques à suivre ou guider la sélection variétale (Hammer *et al.* 2002). Il est intéressant de noter que la modélisation utilisée à des fins exploratoires a montré qu'elle pouvait amener à des résultats contre-intuitifs (Loomis *et al.* 1979) ou qu'elle pouvait permettre de répondre à des questions sur les meilleures stratégies adaptatives des plantes cultivées pour lesquelles l'expert ne pouvait répondre aisément (Hammer *et al.* 2002).

Rapidement la nécessité de pouvoir comparer des modèles et échanger des éléments de modèles entre modélisateur a été identifiée (Loomis *et al.* 1979). Différentes communautés de

[5] La croissance potentielle désigne la croissance de la plante dans le contexte d'une disponibilité parfaite en eau et en nutriments.

chercheurs ont abordé ce sujet sous l'angle de la modularité (Jones *et al.* 2001). L'évolution des techniques informatiques a fortement influencé les outils mis en place par la communauté de la modélisation en agronomie, que ce soit :

- par les approches génériques telles que EPIC (Williams *et al.* 1984) ou STICS (Brisson *et al.* 2009) ;
- par les approches composant (Hillyer *et al.* 2003) ;
- par la mise en œuvre d'ontologie (Beck *et al.* 2010).

En parallèle à la modélisation « grande culture », des démarches de recherche ont été conduites visant à reproduire l'architecture des plantes indépendamment de toute considération physiologique. Le langage formel, défini en 1968 par Aristid Lindenmayer (Lindenmayer 1968), est connu sous le nom de L-System. Son utilisation reste d'actualité et a permis d'obtenir par la simulation des représentations en trois dimensions de plantes virtuelles dont l'aspect est particulièrement proche d'une plante réelle (voir pour exemple la figure 1.2.3). Plus d'éléments sur les L-System et leur utilisation sont accessibles dans le livre « The Algorithmic Beauty of Plants »[6] (Pruzinkiewicz et Lindenmayer 2004).

Une convergence s'est opérée, par la suite, entre les représentations architecturées et la prise en compte des processus biophysiques amenant à la croissance et au développement des plantes. Cette convergence a abouti à une famille de modèles appelés FSPMs (Functional Structural Plant Models) ou modèles structure-fonction. Ces modèles sont de puissants outils de recherche et permettent de tester des hypothèses sur les relations entre les différents organes de la plante (Drouet et Pagès 2007; Vos *et al.* 2007), ou encore, d'identifier les caractères à sélectionner pour obtenir une variété plus performante tel que suggéré par (Vos *et al.* 2010). La frontière entre les FSPMs et les modèles grande culture, intégrant une représentation topologique comme SIMCOT, est assez ténue. C'est la prise en compte explicite des processus au niveau des organes avec une considération du voisinage de ces organes, donc avec une complexité bien plus importante de la représentation des processus, qui fera sortir le modèle de la catégorie des modèles grande culture pour entrer dans celle des FSPMs.

[6] Librement accessible en ligne : http://algorithmicbotany.org/papers/

Figure 1.2.3 : Inflorescence de Lilas construite à l'aide d'un L-System (Pruzinkiewicz et Lindenmayer 2004) p. 92

Le lecteur attentif aura noté la plus grande part accordée à la modélisation grande culture dans cet historique. Comme annoncé dans l'introduction, l'activité d'ITK se focalise sur l'aide à la décision pour les exploitants agricoles en cours de saison. Or, les FSPMs restent, pour l'heure, inadaptés à ce type d'application, c'est pourquoi ils sont exclus du contexte de l'étude. Afin de mieux cerner ce contexte nous allons à présent aborder de manière plus précise la notion de modèle grande culture.

2.3- La modélisation grande culture

Selon Sinclair et Seligman (Sinclair et Seligman 2000), les modèles grandes cultures utilisent les possibilités de l'intégration numérique pour décrire la complexité des comportements des cultures, ceci bien au-delà des capacités des fonctions statistiques. Daniel Wallach livre une description complète de ce type de modèle dans (Wallach 2006), les équations présentées ci-après sont issues de cette publication. Les modèles grande culture sont des modèles

mathématiques décrivant la croissance et le développement d'une culture en interaction avec le sol. Deux points de vue différents sont exposés par l'auteur, ces deux points de vue sont adoptés par le modélisateur en fonction de l'utilisation du modèle qu'il souhaite avoir. Se retrouve donc ici la notion d'intention du modélisateur qui est essentielle et, ce, quel que soit le domaine de modélisation abordé.

Dans le premier cas, le modélisateur se focalise sur l'explication du comportement du système en considérant un modèle à système dynamique. C'est-à-dire qu'il le décompose en un ensemble d'équations (Eq.1) (différentielles ou aux différences) permettant de calculer l'évolution au cours du temps de variables d'état en fonction de paramètres et de variables explicatives (les entrées du système : conditions pédoclimatiques au cours du temps et les conditions initiales du système).

$$U_1(t+\Delta t) = U_1(t) + g_1\,[U(t), X(t)\,;\,\theta]$$

$$\ldots \qquad (1)$$

$$U_S(t+\Delta t) = U_S(t) + g_S\,[U(t), X(t)\,;\,\theta]$$

Ici le temps est désigné par t, Δt désigne l'incrément du temps. Le vecteur $U(t) = [U_1(t), \ldots, U_S(t)]$ contient l'ensemble des variables d'état du système à l'instant t. $X(t)$ est le vecteur des variables explicatives et θ le vecteur des paramètres du système dynamique.

Chaque équation représente un processus biophysique, éventuellement, ce processus peut être divisé en sous-processus explicatifs, le niveau de décomposition le plus bas reposant toujours sur une relation empirique (Amthor 2000).

Dans le deuxième cas, le modélisateur considère uniquement les sorties qui présentent un intérêt pour lui, il s'intéresse alors au comportement du système. Le modèle est alors appelé modèle de réponse, il fournit des variables de réponse qui sont des valeurs de variables d'état du système dynamique à un instant donné ou le résultat de transformation de ces variables d'état. En procédant par intégration au cours du temps, la valeur d'une variable d'état, à un temps T, peut s'écrire en fonction des variables explicatives du modèle depuis t=0 jusqu'à T (Eq.2). Ce qui signifie qu'une variable de réponse Y peut s'écrire comme étant une fonction des variables explicatives et des paramètres (Eq.3).

$$U_i(T) = f_{i,T}\,[X(0), X(\Delta t), X(2\Delta t), \ldots, X(T-\Delta t); \theta] \qquad i = 1, \ldots, S \qquad (2)$$

$$Y = f(X ; \theta) \qquad (3)$$

Cette fonction résulte de l'intégration de fonctions du système dynamique. Bien souvent, il est impossible de la formaliser. C'est l'intégration numérique du système dynamique qui permet d'obtenir la réponse. Il n'est pas de construction de modèle de réponse sans avoir, au préalable, conçu le modèle à système dynamique. Il n'est par contre pas indispensable de construire le modèle de réponse, c'est le cas si le modélisateur souhaite utiliser son modèle à des fins heuristiques, ce qui fut le cas, par exemple, pour le modèle de Turing présenté auparavant.

Cette présentation des modèles grande culture est simplifiée, tel qu'il est formalisé par Wallach, le système dynamique ne permet pas d'exprimer facilement au travers de variables d'état et d'équations la phénologie de la plante. Comme nous l'avons vu précédemment, le développement de la plante induit des changements dans le fonctionnement de ses organes et, par conséquent, la manière dont les processus physiologiques interviennent dans le modèle. Le passage d'un stade phénologique à un autre est, le plus souvent, déterminé par le franchissement d'une valeur seuil qui correspond au cumul de température journalière. Le nouveau stade ayant un impact potentiel sur la formalisation des processus contenus dans le modèle, le modélisateur fait donc appel à des expressions conditionnelles dans la définition de son modèle.

De plus, certaines catégories de modèles grande culture seront difficiles à décrire en utilisant ce formalisme. En effet, au cours de leur cycle de vie, les plantes mettent en place de nouveaux organes, par exemple des feuilles. Chaque feuille possède sa propre surface, la surface foliaire est un déterminant important de la quantité de lumière interceptée par la plante et, donc, de la quantité de matière organique produite. Si un modèle prend en compte de manière explicite chaque feuille et sa surface, la description du système plante devient plus complexe à formaliser. En effet, la structure même du système devient dynamique, de nouvelles variables d'état apparaissent au cours du temps de la simulation. Tous les modèles grande culture ne sont pas concernés par cette problématique. Certains d'entre eux n'ont qu'une description très basique de la plante, celle-ci est considérée comme une seule grande feuille, d'où l'appellation de modèles *big leaf*, le plus connu d'entre eux étant sans doute le

modèle STICS dont Nadine Brisson était à l'origine. Dans ces modèles, tous les organes (feuille, grain, interface plante-sol) sont en place au début de la simulation, il ne s'agit plus que d'un système dynamique sans problématique concernant la dynamique de sa structure.

Nous avons pu identifier deux autres catégories de modèles grande culture pour lesquels la dynamique de la structure est prise en compte : les modèles à strates (*leaf-layered models*) et les modèles à topologie explicite. Les modèles à strates tiennent leur nom de la décomposition du couvert qui y est faite. La plante est subdivisée en plusieurs horizons, plusieurs strates, chacune d'entre elles regroupant les feuilles ayant le même âge. Toutes les feuilles de même âge sont considérées comme une seule grande feuille (à la manière d'un modèle *big leaf*). L'intérêt, par rapport à un modèle *big leaf*, est de permettre de représenter plus finement la dynamique des surfaces foliaires, c'est le cas par exemple de Sirius pour la modélisation du blé (Jamieson *et al.* 1998). Quant à eux, les modèles à topologie explicite intègrent les organes de manière individualisée. Ramifications, feuilles et fruits sont créés au cours de la simulation. Se trouve, parmi les cas d'application, la mise en œuvre de processus stochastiques permettant de rendre compte de la chute de certains organes en cas de stress comme par exemple dans Cotton (Jallas 1998). Comme évoqué précédemment, la frontière entre ce type d'approche et un modèle FSPM est assez ténue, il serait possible d'argumenter que Cotton fait partie des FSPM et non des modèles grande culture.

Outre la représentation de la structure, il semble important de spécifier les échelles temporelles desquelles dépendent les modèles grandes cultures. L'échelle est fortement influencée par la nature circadienne de la croissance de la plante, dès lors, une simulation d'un modèle grande culture se déroule quasi exclusivement de date à date, c'est-à-dire suivant un pas de temps journalier. Les modèles conçus pour représenter la dynamique d'une plante de plein champ (à l'exclusion des expériences en milieu contrôlé) sont contraints par les mesures qui peuvent y être effectuées. Les conditions environnementales - essentiellement les conditions climatiques - gouvernent la dynamique du système plante-sol. Ces données sont obtenues grâce à des stations météorologiques suivant les cas avec une fréquence journalière ou horaire. C'est pourquoi il est rare de représenter des processus à un pas de temps infra-horaire.

Après cet aperçu de la modélisation grande culture, nous allons présenter un des modèles représentant une parcelle viticole mis en place par ITK ainsi que son contexte d'utilisation. Il s'agit d'un des modèles ayant servi de cas d'étude pour le travail de thèse dont fait l'objet ce mémoire.

3- Les modèles mécanistes mis en place chez ITK

Les modèles produits par ITK ont, dans la majeure partie des cas, pour objectif d'être intégrés dans des outils d'aide la décision. Ces outils peuvent s'adresser directement à l'exploitant agricole ou à des personnels techniques intervenant sur les parcelles de différents exploitants. Parmi les finalités possibles de ces outils se trouvent la gestion du statut hydrique de la culture, l'état phytosanitaire de la culture ou encore la prévision du rendement. Le modèle vigne a été défini dans l'objectif d'optimiser l'irrigation de la vigne.

3.1- Le modèle vigne

Le modèle présenté dans cette partie n'est qu'une des déclinaisons existant pour le modèle vigne. S'agissant d'une activité de recherche, plusieurs configurations ont été testées. Nous nous contenterons de la déclinaison la plus simple pour introduire le sujet. Dans un premier temps, nous donnons les éléments pris en compte dans ce modèle avec les hypothèses qui les accompagnent. Nous aborderons ensuite la définition conceptuelle de ce modèle et enfin nous présenterons quelques unes des équations régissant son comportement.

3.1.1- Identification des éléments d'intérêt pour le modèle vigne

L'objectif de l'outil d'aide à la décision est de se focaliser sur le statut hydrique de la parcelle, c'est-à-dire la disponibilité en eau dans le sol par rapport à la quantité d'eau nécessaire pour assurer normalement la perte d'eau par évaporation (interface sol-atmosphère) et par transpiration (interface plante-atmosphère). L'aide à la décision se concentre sur le maintien d'un stress hydrique de la vigne à certains stades de son développement en fonction de la qualité du vin que le viticulteur souhaite produire. La figure 1.3.1 représente les éléments d'importance pour pouvoir représenter le statut hydrique de la culture.

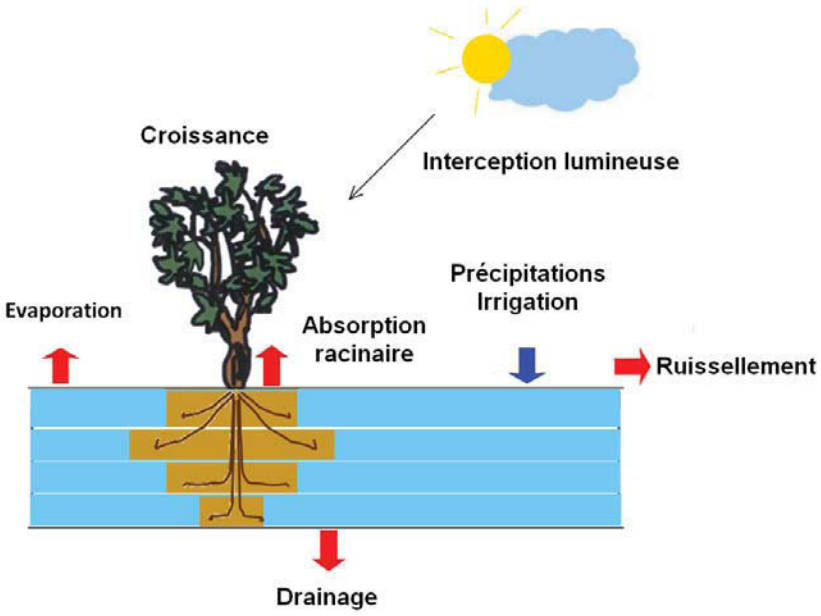


Figure 1.3.1 : Représentation schématique des différents phénomènes pris en compte dans le modèle vigne

L'objectif du modèle impose la prise en compte des phénomènes importants aboutissant à un apport ou une perte d'eau pour le système sol-plante. Les apports proviennent des précipitations ou de l'irrigation décidée par le viticulteur. Les pertes sont issues du drainage et du ruissellement d'une part et de la vaporisation de l'eau d'autre part. Cette vaporisation se produit par évaporation au niveau du sol et transpiration de la plante. La transpiration de la plante se produit presque exclusivement par les feuilles, la surface foliaire est donc importante à prendre en compte. L'interception lumineuse par le couvert végétal gouverne à la fois la croissance de la plante et sa transpiration. Il n'y a, par contre, pas d'objectif de rendement concernant ce modèle, ceci permet d'avoir une représentation simple du couvert végétal tout en conservant une bonne estimation de la transpiration.

3.1.2- Conceptualisation du modèle

La démarche de conceptualisation d'un modèle est réalisée après identification des éléments d'intérêt. Elle conduit à la définition explicite des processus pris en compte, leurs relations et leur intervention au cours de la simulation. La construction d'un modèle conceptuel préalable à la réalisation du modèle est une constante dans beaucoup de domaines de la M&S, elle reste cependant le parent pauvre de ce domaine, en effet, peu d'activités de recherche et aucun outil informatique ne lui sont dédiés (Robinson 2007). Le formalisme retenu est proche des diagrammes de flux (*flowcharts*). Il correspond à la pratique adoptée par les modélisateurs

d'ITK et, plus généralement, par beaucoup de modélisateurs en agronomie comme, par exemple, (Wang et Engel 2000; Lawless *et al.* 2005; Willocquet *et al.* 2008; Hakojarvi *et al.* 2010). L'absence d'outils formels pour la définition du modèle conceptuel conduit les modélisateurs à ne le réaliser que sur papier. Cette pratique aboutit à une construction informelle, certains éléments, bien que présents à l'esprit du modélisateur, ne sont pas explicités :

- les données nécessaires au fonctionnement du processus appelées entrées ;
- les variables d'état qu'il exploite ;
- parfois même, le pas de temps correspondant au processus ;
- la structure de données décrivant le système plante-sol.

La figure 1.3.2.A est une représentation simplifiée du modèle conceptuel pour la vigne. Les différents processus que le modélisateur souhaite prendre en compte suite à l'identification des besoins sont représentés dans les encadrés noirs, les flèches qui les relient les uns aux autres traduisent la nature séquentielle des modèles grande culture. Au cours d'un pas de temps de simulation, l'interception lumineuse est calculée avant que ne soit exécuté le modèle VAT (Vegetation Atmosphere Transfer). Ce modèle est particulier, il dépend des données disponibles et commence donc par un test (1) de la présence ou non de l'ETP[7] (EvapoTranspiration Potentielle) dans les données d'entrée. Toutes les stations météorologiques ne sont pas à même de fournir cette quantité. En son absence, il est nécessaire de la calculer à partir d'autres données d'entrée (2). Une fois l'ETP obtenue, il est possible de procéder au calcul de l'évapotranspiration réelle, c'est-à-dire la perte d'eau effective du système plante-sol au cours de la journée. Notons, ici, que la décomposition du modèle VAT correspond à une hiérarchisation du modèle, le processus d'échange entre la plante et l'atmosphère est décomposé en sous processus. Une fois l'évapotranspiration réelle calculée, le modèle sol permet de faire le bilan hydrique du compartiment sol en prenant en compte les différents apports et pertes d'eau. Enfin, le développement de la plante est pris en compte (modèle phénologique) puis sa croissance est déterminée.

L'identification des éléments d'intérêt pour le modèle a conduit à retenir une représentation simplifiée du couvert végétal (3). Un rang de vigne est assimilé à un parallélépipède

[7] L'ETP correspond à la quantité d'eau qui est perdue par le système sol-plante au profit de l'atmosphère lorsque la disponibilité en eau n'est pas limitée. Elle dépend essentiellement de la température, du rayonnement, du vent et de l'humidité relative de l'air.

rectangle. Il se caractérise par une hauteur, une largeur et une porosité. La longueur du rang est implicitement considérée infinie, ceci permet de s'affranchir des effets de bord. La signification de la porosité fait l'objet de controverse au sein de la communauté des modélisateurs. Nous nous contenterons de la définir comme la proportion de lumière incidente que le couvert n'intercepte pas, cette grandeur vaut 1 en début de saison et 0 lorsque le couvert atteint sa taille maximale.

Le modélisateur a également choisi de représenter l'influence qu'ont les rangs de vigne sur l'interception lumineuse de leurs voisins. En effet, un rang peut faire de l'ombre à son voisin réduisant ainsi la quantité de lumière que ce dernier reçoit. L'orientation des rangs de vignes devient alors importante à prendre en compte tout comme l'évolution de la position du soleil au cours de la journée. Ceci a conduit à intégrer un processus calculant la position du soleil (4) ainsi que le passage du modèle d'interception au pas de temps horaire (5).

La figure 1.3.2.B définit la succession de stades phénologiques considérée dans la simulation, pour chaque stade la condition d'avancement est définie :

- date observée fournie par l'utilisateur ;
- somme de températures : si la température moyenne du jour (Tmean) est supérieure à Tb, la quantité Tmean - Tb est ajoutée à un cumul. Une fois le seuil franchi, le stade suivant est atteint ;
- équations de Chuiné et Pisek : elles prennent en compte la température moyenne journalière. Le résultat de l'équation est ajouté à un cumul. Une fois le seuil franchi, le stade suivant est atteint.

Les mécanismes régissant le développement de la plante (son évolution phénologique) sont complexes et pilotés par des facteurs multiples. Il est donc difficile de rendre compte mathématiquement de ces mécanismes autrement que de façon empirique, ce qui s'est avéré simple et efficace. La partie suivante aborde de manière explicite certaines représentations mathématiques des différents processus pris en compte dans le modèle vigne.

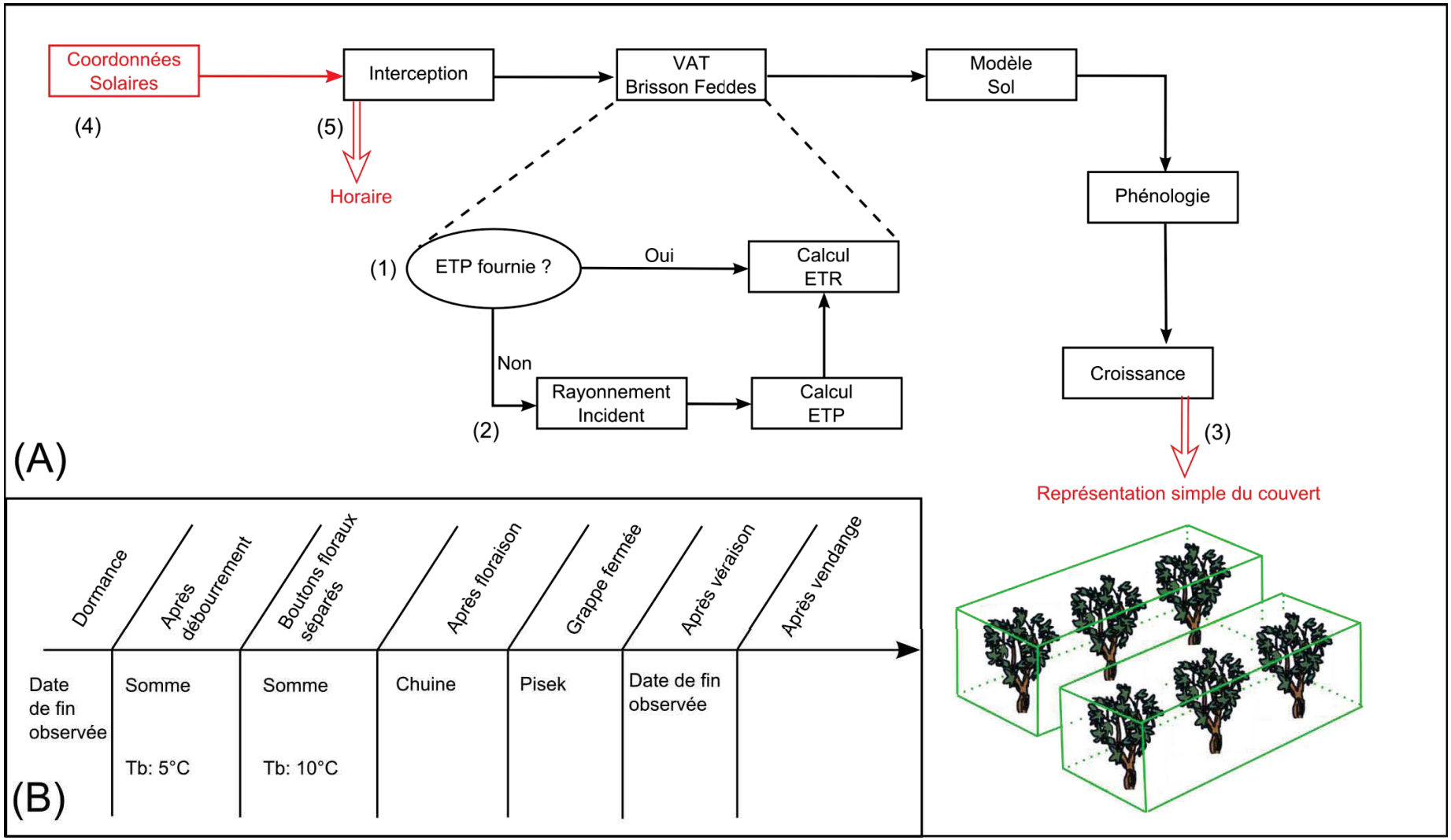


Figure 1.3.2 : Conception du modèle de croissance de la vigne. A) Modèle conceptuel et représentation du couvert. B) Stades phénologiques pris en compte

3.1.3- Les représentations mathématiques du modèle vigne

Dans cette partie, seules seront données les représentations de l'interception lumineuse et d'une des formalisations de la croissance. Plusieurs modèles ont été réalisés pour cette dernière, nous avons retenu le plus simple d'entre eux. Les deux représentations devraient être suffisantes pour décrire les caractéristiques des modèles utilisés par ITK.

La figure 1.3.3 décrit l'évolution de descripteurs du couvert végétal au cours du temps. La courbe A représente la dynamique de la hauteur du couvert, bien que non représentée, la largeur du couvert suit exactement le même principe aux paramètres près. La courbe B correspond à l'évolution de la porosité du couvert. Comme évoqué ci-dessus, elle caractérise la quantité de lumière que le végétal n'intercepte pas, d'où une dynamique opposée à celle de la hauteur et de la largeur du couvert végétal.

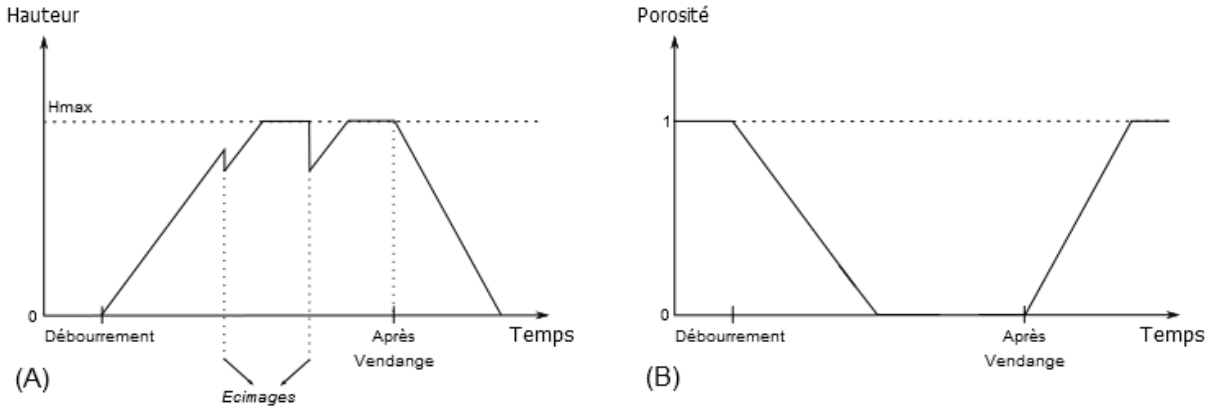


Figure 1.3.3 : Dynamique de la hauteur (A) et de la porosité (B) du couvert de la vigne suivant le modèle de croissance linéaire

L'axe temporel pour ces deux figures met en exergue la dépendance de la croissance vis-à-vis de la phénologie : avec le débourrement des bourgeons, débute la croissance du couvert végétal. Les interventions du viticulteur ont également un impact sur les deux dynamiques. L'écimage et le rognage sont des interventions ponctuelles qui vont lui permettre de réduire respectivement la hauteur et la largeur des rangs de vigne. Enfin, la vendange détermine le début de la sénescence des feuilles et, *in fine*, leur chute.

Le modèle de croissance linéaire suppose un accroissement constant de la taille du couvert à compter du débourrement. Un taux d'accroissement (*gr* : pour growth rate) est paramétré dans le modèle. En phase de croissance normale les équations du système dynamique sont donc assez simples :

$$H(t) = H(t-1) + grH$$

$$L(t) = L(t-1) + grL$$

$$P(t) = P(t-1) + grP$$

Une saturation du couvert est admise pour ce modèle de telle sorte que, par exemple pour la hauteur, le système d'équations soit :

$H(t-1) > Hmax - grH$ $\quad H(t) = Hmax$H(t) = Hmax

$H(t-1) \leq Hmax - grH$ $\quad H(t) = H(t-1) + grH$

De plus, l'intervention du viticulteur, au temps *t*, pour réduire la hauteur du couvert à une valeur H_0, doit pouvoir être prise en compte, tout comme l'influence de la phénologie et de la vendange. Il devient alors difficile de formaliser le processus de manière mathématique, alors que l'algorithme qui lui correspond est relativement aisé à écrire :

```
Si phenologie < debourrement
    Si rognage
        H=H0
    Sinon
        H = min(Hmax, H+grH)
    Fin si
 Sinon si phenologie < vendange
    H = max (0, H+dgrH)
Sinon
   H=0
Fin si
```

Cet exemple montre que la définition par Wallach des systèmes dynamiques est théorique. La forme de la fonction qui détermine la valeur de *H* dépend d'une autre variable d'état qui joue un rôle qualitatif et non quantitatif. Dès lors, la formalisation des équations est difficile à mettre en œuvre dans le cadre de la modélisation grande culture.

Les variables d'état calculées par le modèle de croissance interviennent dans le calcul d'autres processus, c'est le cas de l'évaluation de l'interception lumineuse par le couvert végétal et du

rayonnement effectivement absorbé par la plante. Ce calcul se base sur les équations définies par (Riou *et al.* 1989). L'équation calculant le rayonnement absorbé définie par Riou est la suivante :

$$R_{aL} = (1 - a_{\mathrm{L}})R_{\mathrm{i}} + a_{\mathrm{s}} n_{\mathrm{r}} (1 - a_{\mathrm{L}})(R_{\mathrm{g}} - R_{\mathrm{i}})$$

Avec : a_L : albedo de la feuille R_i : rayonnement intercepté

R_g : rayonnement global a_S : albedo du sol

n_r : fraction du rayonnement réfléchi par le sol et intercepté par la plante

Une série d'équations successives fournit le rayonnement intercepté par la plante (R_i) en prenant en compte les valeurs des grandeurs descriptives du couvert : hauteur, largeur, porosité. Il paraît peu opportun de livrer plus de détails sur les calculs qui sont effectués. Notons que la quantité de radiation absorbée par la plante, que Wallach considère comme une variable d'état, n'a qu'une vocation transitoire dans le modèle de vigne. En effet, elle n'est calculée que pour pouvoir faire le calcul de l'ETR suivant les formules fournies par (Brisson et Perrier 1991) et (Feddes *et al.* 1978). De plus, sa valeur au pas de temps *t* est indépendante de sa valeur au pas de temps *t-1*. Ceci amène les modélisateurs à définir ce type de données comme une sortie du modèle d'interception et une entrée du modèle VAT.

Nous avons souhaité présenter dans cette partie les caractéristiques des modèles employés par ITK en présentant les différentes étapes qui conduisent le modélisateur à :

- identifier les processus qu'il souhaite mettre en jeu dans son modèle ;
- définir les modalités d'intervention de ces processus dans le modèle ainsi que leurs interactions ;
- définir la représentation structurelle du système ;
- identifier les formalismes mathématiques et logiques représentant les processus.

Cette démarche incertaine et itérative s'inscrit dans un contexte industriel de production logicielle. La partie suivante permet de mieux appréhender les conséquences de cette démarche sur la production. Ces dernières forment la problématique centrale du travail de thèse.

4- Conception et implémentation des modèles : une problématique

Les outils d'aide à la décision d'ITK sont orientés web, ils sont développés sur une plateforme Java Entreprise Edition (JEE). Pour avoir une complète maitrise de l'environnement de production, le système expert et le couple simulateur-modèle, sur lequel il repose, sont également développés en Java. Or, les agronomes modélisateurs concevant et évaluant les modèles ne sont pas formés au développement objet. De plus, à notre connaissance, il n'existe pas d'environnement Java permettant d'implémenter un modèle scientifique, du type de ceux que nous avons présentés, et d'en évaluer le comportement. Ces éléments conduisent les modélisateurs à développer un prototype du modèle dans un langage de programmation pour le calcul scientifique : Matlab. La figure 1.4 expose le processus suivi au sein d'ITK pour la réalisation d'un nouveau modèle pour un outil d'aide à la décision, de l'identification des éléments d'intérêt jusqu'à l'obtention du code exploitable sur la plateforme JEE. En premier lieu (1), vient l'identification des éléments d'intérêt par rapport à la problématique ciblée telle qu'évoquée au 3.1.1. Ensuite les modélisateurs effectuent une revue bibliographique (2) sur les différentes représentations possibles des processus retenus en tenant compte :

- de l'espèce cultivée considérée ;
- de la disponibilité de données pour le paramétrage du modèle ;
- de la disponibilité de données pour l'évaluation de ses capacités prédictives.

Cette revue conduit à la définition d'un premier modèle conceptuel (3). Comme exposé au 3.1.2, ce modèle conceptuel reste informel, basé sur un travail papier. L'absence d'outils formels pour la modélisation conceptuelle (Robinson 2007), ainsi que la difficulté d'expliciter les équations du système dynamique (Cf. 3.1.3), ont pour conséquence de faire de l'implémentation du modèle en Matlab (4) la seule référence complète du modèle. Une documentation du modèle en langage naturel est réalisée, cependant la linéarité d'un document texte est peu adaptée à la description d'un modèle qui peut être parcouru en fonction de son flot d'exécution (les processus exécutés successivement au cours de la simulation) ou de flots de données spécifiques (de la sortie d'un processus aux entrées des différents processus qui peuvent l'exploiter).

Ce prototype est testé (5), de ces tests résulte un processus itératif (6) de nouvelle recherche bibliographique, conceptualisation, implémentation et tests ou du passage à l'implémentation en Java (7) qui est ensuite validée (8).

Ce processus est théorique, dans la pratique et compte tenu du temps nécessaire à stabiliser le modèle scientifique, les délais de livraison peuvent contraindre l'entreprise à débuter le développement Java avant que le modèle ne soit achevé. Cette contrainte temporelle a plusieurs répercussions dont les conséquences peuvent être dommageables.

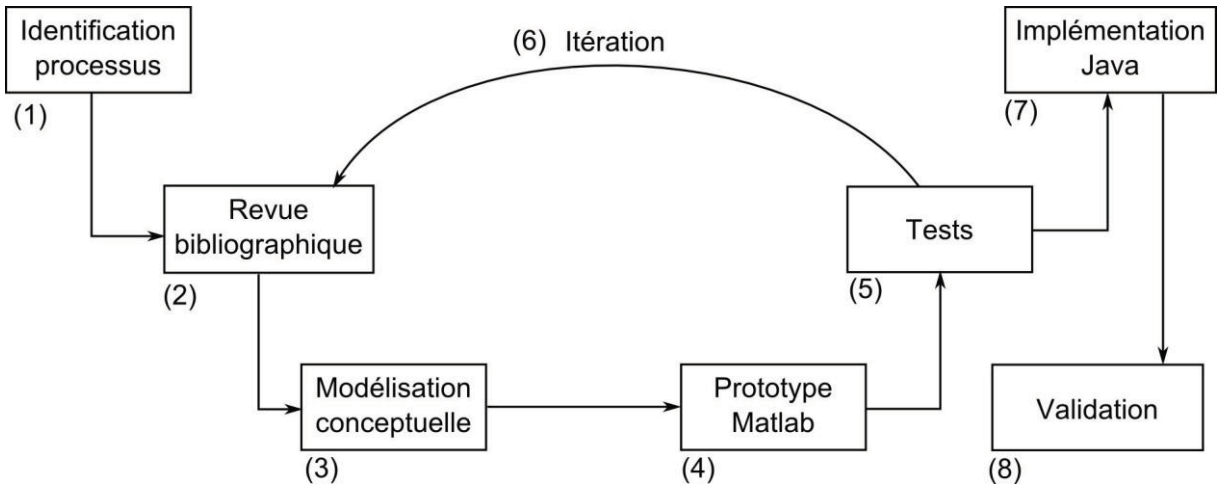


Figure 1.4 : Processus de réalisation, par l'équipe d'ITK, d'un nouveau modèle : de l'identification des processus à intégrer jusqu'à l'obtention de l'implémentation Java valide.

Au cours des itérations, le modélisateur peut chercher à gagner du temps en omettant de mettre à jour la documentation du modèle et en modifiant directement le prototype. Une disjonction s'opère alors entre le document et le modèle Matlab, cette disjonction peut être problématique pour la reprise ultérieure du modèle par un autre modélisateur ou par un informaticien pour la transcription en Java. Le développement en parallèle des deux implémentations du modèle favorise les risques de disjonction inhérents à l'opération de transcription. Ces risques s'ajoutent à une perte en termes de productivité. En effet, la double implémentation du modèle et sa double validation sont la répétition d'activités similaires, les deux implémentations ne diffèrent que par le langage utilisé pour les exprimer. En un certain sens, la modélisation conceptuelle est elle-même similaire aux implémentations du modèle. Celle-ci permet de définir les entrées, sorties et le contenu du modèle (Robinson 2008), ce sont les éléments nécessaires et suffisants à la définition du modèle et donc à son implémentation. (van der Zee *et al.* 2011) soulignent la difficulté de distinguer la limite entre la modélisation conceptuelle et l'implémentation du modèle. A notre sens, la modélisation conceptuelle a pour objectif de passer sous silence les détails techniques de l'implémentation.

Ces détails sont à la fois associés à l'environnement technique qui sera utilisé (*p.ex.* langage objet ou procédural) et à des invariants qui constituent la modalité d'exécution de la simulation (*p.ex.* la façon dont l'écoulement du temps est pris en compte). Cette dernière catégorie correspond aux pratiques d'une communauté d'intérêt telle que s'y réfèrent (Balci *et al.* 2008). Dans cette publication, Balci et ses coauteurs définissent le modèle conceptuel comme un ensemble de constructions spécifiques du plus haut niveau d'abstraction (*p.ex.* diagrammes, dessins et équations), ces constructions permettent à la communauté de traiter de problèmes spécifiques à son domaine d'intérêt. Ce point de vue rejoint celui de (Lacy *et al.* 2001) définissant une catégorie de modèles conceptuels orientés domaine par opposition aux modèles conceptuels orientés conception de la simulation, ces derniers relevant plutôt, selon nous, de l'ingénierie logiciel.

Si, dans un domaine d'intérêt donné, le modèle conceptuel associé aux connaissances de mise en œuvre du modèle sont suffisants pour obtenir l'implémentation du modèle, serait-il possible de proposer un environnement de modélisation conceptuelle permettant aux modélisateurs d'ITK de formaliser leurs modèles par la manipulation de concepts à un haut niveau d'abstraction ? Serait-il possible d'obtenir automatiquement, à partir du modèle conceptuel, l'implémentation Java du modèle sans l'intervention d'ingénieurs logiciel ? Une telle solution permettrait à la fois de couvrir la problématique de la documentation du modèle et d'éviter les risques encourus lors de la transcription du modèle. De plus, si ce modèle comporte l'ensemble des informations nécessaires à la réalisation de la simulation dans le cadre de la modélisation grande culture, ne serait-il pas envisageable de procéder à la simulation dans l'environnement de modélisation conceptuelle ? Ceci permettrait de s'affranchir de la phase de prototypage en Matlab et mènerait donc à une amélioration de la productivité des équipes d'ITK. Le modèle conceptuel prendrait la place du prototype, il devrait donc être validé, tout comme l'implémentation Java. A l'instar de l'obtention de l'implémentation Java, ne serait-il pas possible d'obtenir à partir des scénarios utilisés pour la validation du prototype les scénarios de test permettant de valider l'implémentation Java ?

Si la démarche se focalise sur la productivité, il paraît important d'évoquer la réutilisabilité de certaines parties des modèles. En effet, certaines représentations de processus, comme la loi de Beer-Lambert pour l'absorption lumineuse par le couvert végétal, sont couramment utilisées. Doter l'environnement de modélisation conceptuelle d'un mécanisme permettant de réutiliser des éléments d'autres modèles sans avoir besoin de les spécifier de nouveau, présenterait donc un intérêt non négligeable.

Pour répondre à ces questions le travail de thèse s'est orienté sur les possibilités offertes par l'Ingénierie Dirigée par les Modèles (IDM) et plus particulièrement les DSLs (Domain-Specific Languages). Plusieurs éléments appuient ce choix, par rapport aux attentes sur les modèles conceptuels et celles concernant l'environnement conception, objectif de ce travail. En effet, les DSLs permettent d'élever le niveau d'abstraction pour un domaine précis par l'utilisation d'une syntaxe textuelle ou graphique (Greenfield et Short 2003), rejoignant ainsi la définition du modèle conceptuel offerte par (Balci *et al.* 2008). De plus selon (Booch *et al.* 2004), les DSLs font des suppositions sur les pratiques du domaine réduisant ainsi la taille de la spécification de l'application. Ceci rejoint les usages du domaine que nous avions qualifiés d'invariants pour la modélisation. Enfin, toujours selon (Booch *et al.* 2004), les DSLs offrent la possibilité de générer du code dans un langage de programmation généraliste mais également d'exécuter directement le modèle défini, ce qui permettrait de répondre aux attentes définies pour l'environnement de modélisation conceptuelle.

5- Conclusion

Dans ce chapitre, après avoir discuté la notion de modèle, nous avons exposé quelques éléments de physiologie végétale nécessaires à la compréhension du domaine de la modélisation en agronomie. Après avoir dressé un bref historique de ce domaine de la modélisation, nous avons détaillé les caractéristiques des modèles grande culture qui correspondent aux modèles mis en œuvre par la société ITK. Ceux-ci lui permettent de produire des outils d'aide à la décision. Le processus industriel mis en en place pour y parvenir fait intervenir une étape de modélisation conceptuelle informelle avant que les modélisateurs ne construisent un prototype du modèle grande culture dans un environnement Matlab.

Avant que le prototype n'ait atteint sa forme définitive, le modèle est implémenté en code Java par des ingénieurs en génie logiciel, afin de pouvoir être intégré dans l'outil d'aide à la décision en cours de développement. La double implémentation - prototype en Matlab et modèle en Java – a un coût en termes de productivité, la seule différence entre ces deux implémentations étant en théorie limitée aux langages employés. Cependant, des différences peuvent également apparaître suite aux opérations de traduction du prototype ainsi que lors de l'évolution du prototype. L'opération de transcription du Matlab vers le Java est en effet sujette à erreur. D'autant plus que le modèle conceptuel devient rapidement obsolète avec

l'évolution du prototype conduisant alors à ce que l'implémentation de celui-ci soit la seule référence fiable au modèle.

Compte tenu de ces éléments, nous avons souhaité orienter notre travail de thèse sur cette problématique de production spécifique à la société ITK. Pour parvenir à y répondre, nous avons suggéré l'emploi des techniques issues de l'IDM afin de mettre en place un DSL dédié à la modélisation grande culture. Ce DSL a pour objectif de permettre la modélisation conceptuelle formelle des modèles grande culture à partir de laquelle il serait possible d'obtenir, de manière automatique, l'implémentation en Java du modèle.

Afin de mieux percevoir les potentialités de l'IDM et des DSLs, le second chapitre de ce mémoire présente une définition de l'IDM, de l'ensemble des techniques qu'elle regroupe et des perspectives qu'elle offre pour ce travail de thèse. Par la suite, un ensemble d'outils issus de l'IDM ou du monde de la modélisation et de la simulation sera présenté. Nous nous attacherons à montrer leur intérêt ainsi que les limites qu'ils présentent quant aux objectifs fixés dans le cadre de cette thèse.

Chapitre 2 – Présentation de l'IDM et de différents outils de modélisation et simulation

Ce chapitre se structure en deux parties. La première vise à présenter de l'Ingénierie Dirigée par les Modèles (IDM). Il serait difficile de se prétendre exhaustif sur le sujet, l'essor considérable qu'a connu l'IDM depuis 2000 a donné lieu à nombre d'initiatives et de techniques différentes. Cependant cette présentation devrait être suffisamment complète pour comprendre les concepts et enjeux couverts par l'IDM dans le cadre de la mise en place d'applications de simulation. La seconde partie, quant à elle, permet d'appréhender différents outils existant dans le domaine de la modélisation et de la simulation (M&S) en se focalisant sur leurs points forts et également leurs limites vis-à-vis de la problématique de cette thèse.

1- L'ingénierie dirigée par les modèles

L'IDM[8] est une approche d'ingénierie logiciel récente, elle est née du constat que le paradigme objet a atteint ses limites à la fin du siècle dernier (Greenfield et Short 2003). La production logiciel reposait, et repose encore, trop sur la manufacture du code. Les coûts de production conséquents et le manque de main d'œuvre qualifiée ont fait naître l'idée de l'industrialisation de la production logiciel. C'est l'industrie logiciel qui en est à l'initiative avec la proposition de la spécification du MDA (Model-Driven Architecture) par l'Object Management Group (OMG 2001). Rapidement, le monde de la recherche s'y est intéressé (Bézivin et Gerbé 2001; deLara et Vangheluwe 2002b; Kent 2002), avec pour objectif d'en mieux formaliser les concepts et d'éviter les écueils nés d'une définition parfois floue et trop

[8] En anglais, Model-Driven Engineering (MDE).

complexe (Favre 2004b). C'est donc à la convergence entre l'industrie et le monde de la recherche académique que l'IDM trouve son origine.

L'IDM a pour objectif de couvrir l'ensemble du cycle de vie applicatif de la conception à la maintenance ou l'évolution de l'application. Pour ce faire, elle fédère un ensemble de techniques préexistantes, parmi lesquelles la rétro-ingénierie, la génération de code ou encore les transformations, autour d'un paradigme : tout est modèle. Si ces techniques ne sont pas nouvelles, l'IDM leur a apporté un cadre formel unificateur. De même, l'utilisation de modèles n'est pas nouvelle, la méthode Merise ou encore l'Unified Modeling Language (UML) permettaient déjà de produire des modèles au cours de la phase de conception (Favre *et al.* 2006b). Cependant, l'IDM a permis de passer de modèles « contemplatifs » à des modèles productifs, offrant ainsi la possibilité de s'affranchir des approches centrées sur le code qui conduisent au code comme référence et à une absence de mise à jour de la documentation (Favre 2004b). Ceci rejoint la problématique de la thèse concernant la réalisation du modèle conceptuel ainsi que le passage à un prototype Matlab qui devient la référence du modèle scientifique concerné.

Selon Jean Bézivin (Bézivin 2005b), si le paradigme de la programmation orientée objet, « tout est objet », a permis de réaliser des efforts importants en terme de généricité et de capacité d'intégration, celui associé à l'IDM, « tout est modèle », devrait aboutir à l'industrialisation de la production logiciel. Pour ce faire, elle s'appuie sur les trois relations de base : *ReprésentationDe*, *ConformeA* et *TransforméEn*. Ces relations font partie des concepts fondamentaux de l'IDM.

1.1- Concepts fondamentaux de l'IDM

1.1.1- Les modèles et la relation *ReprésentationDe*

La notion de modèle a été abordée dans le premier chapitre de ce mémoire ; au sein même de la communauté de l'IDM, de nombreuses définitions sont proposées. S'il n'existe pas de définition universelle du terme « modèle » (Favre *et al.* 2006b; Kühne 2006), un consensus se dégage tout de même sur ses objectifs et son utilisation. L'une des définitions les plus concises et complètes, trouvées au sein de la communauté, est fournie par Jean Bézivin et Olivier Gerbé (Bézivin et Gerbé 2001) :

« A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system ».

Comme exposé dans (Barbier *et al.* 2011), nous avons souhaité que cette définition soit complétée par la restriction suivante « in the restricted scope of the chosen simplification ».

A partir de cet élément central qu'est le modèle, l'IDM définit une première relation qui relie le système au modèle, il s'agit de la relation « *ReprésentationDe* » (Atkinson et Kühne 2003; Seidewitz 2003; Bézivin 2004) notée μ. Bien que non formalisée auparavant, cette relation n'est pas nouvelle en ingénierie logiciel, elle existe par exemple dans la méthode Merise ou le langage UML. Comme écrit précédemment, la force de l'IDM repose sur sa capacité à rendre les modèles productifs. Pour y parvenir, une démarche IDM se focalise sur un point de vue précis, un aspect, de l'applicatif à mettre en place. Cet aspect peut être technique, comme les fonctionnalités de traces ou la gestion de la persistance, ou encore être focalisé sur les concepts du métier pour lequel l'application est conçue, comme la modélisation et simulation grande culture.

La restriction du domaine permet de définir un ensemble de concepts de haut niveau d'abstraction et les relations entre ces différents concepts. Ceci conduit à l'obtention d'un langage expressif, directement utilisable par les spécialistes du domaine, ce langage s'affranchit des détails techniques de l'implémentation. L'ensemble des concepts et règles d'association définissant ce langage correspond à un métamodèle du domaine.

1.1.2- Le Métamodèle et la relation *ConformeA*

La notion de métamodèle comme celle de modèle sont l'objet de discussion au sein de la communauté IDM. Se trouve couramment la définition selon laquelle un métamodèle est un modèle d'un modèle (Steinberg *et al.* 2008) (p.34). Cette conception, trompeuse (Favre 2004b), du métamodèle est issue des premières heures de la MDA. Le métamodèle est la structure de base autour de laquelle s'articule une démarche IDM. C'est grâce à lui que les modèles, exprimés dans le langage qu'il décrit, vont obtenir un cadre formel. Il serait délicat d'adopter une démarche IDM sans une bonne compréhension de la notion de métamodèle.

La figure 2.1.1 est issue de (Favre *et al.* 2006b), elle propose de considérer la représentation du territoire français par un modèle prenant en compte sa décomposition en zones administratives. Régions et départements sont les deux concepts que le cartographe cherche à retranscrire. Ces concepts sont portés par la légende qui forme le métamodèle de la carte administrative. Toute carte administrative est *ConformeA* sa légende, la relation de conformité est notée χ.

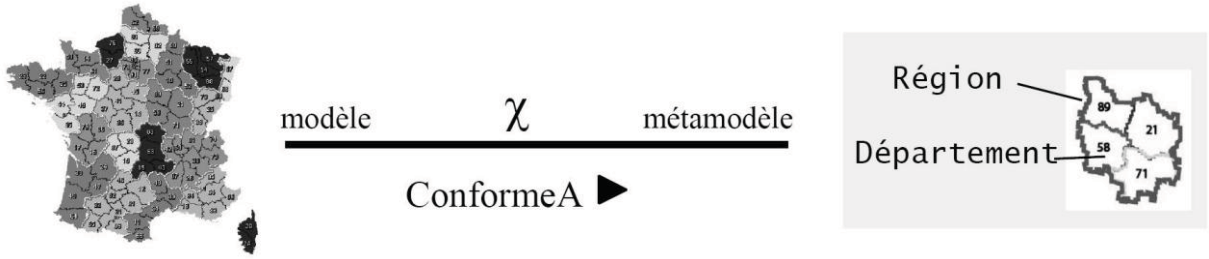


Figure 2.1.1 : Relation de conformité suivant l'exemple des cartes, tiré de (Favre *et al.* 2006b)

Cette figure fait également apparaître une notion importante, le modèle appartient à l'ensemble des cartes qu'il serait possible de formaliser en utilisant le métamodèle. Il s'agit de l'ensemble des phrases syntaxiquement correctes qu'un cartographe pourrait réaliser. Cet ensemble correspond au langage des cartes administratives. Le métamodèle tient lieu de modèle de ce langage (Favre 2004c). En effet, il est analogue à une grammaire, la théorie des langages a établi qu'une grammaire est un modèle du langage. Ceci explique la définition proposée par l'Object Management Group, dans le glossaire de sa spécification du Meta-Object Facility (v. 1.4.2) (OMG 2002) :

"Metamodel: A model that defines the language for expressing a model."

Comme le souligne Favre (Favre 2004a), en considérant l'ensemble des modèles d'un langage de modélisation, la définition suivante est donc correcte :

« A metamodel is a model of models »

Mais elle pose problème, une lecture rapide ou une mauvaise écoute peuvent amener à l'omission du « s » terminal et aboutir à la définition donnée par (Steinberg *et al.* 2008). Ceci est sans doute responsable de la mauvaise compréhension de ce qu'est un métamodèle. C'est pourquoi nous lui préférons la définition fournie par Seidewitz (Seidewitz 2003) :

« *A metamodel makes statements about what can be expressed in the valid models of a certain modeling language* »

La notion de validité, à laquelle Seidewitz fait appel, traduit bien le concept de conformité du modèle vis-à-vis de son métamodèle et la relation entre le métamodèle et le langage de modélisation.

Si le métamodèle est un modèle, il est en toute logique possible de définir le métamodèle auquel il doit se conformer. Il s'agit alors d'un méta-métamodèle. En reprenant la définition de Seidewitz, on obtient alors :

« A meta-metamodel makes statements about what can be expressed in the valid metamodels of a certain metamodeling language »

Le méta-métamodèle étant un métamodèle, il est possible de le décrire en utilisant le langage de métamodélisation qu'il définit, il est alors conforme à lui-même. Il s'agit du principe de métacircularité.

La figure 2.1.2 est une représentation pyramidale, couramment employée dans la communauté IDM, des différents niveaux d'abstraction et de leur dénomination. Son socle, le niveau M_0, est réservé aux systèmes modélisés. Le niveau M_1 correspond au premier niveau d'abstraction : les modèles. Les niveaux M_2 et M_3 désignent respectivement l'ensemble des métamodèles et le méta-métamodèle.

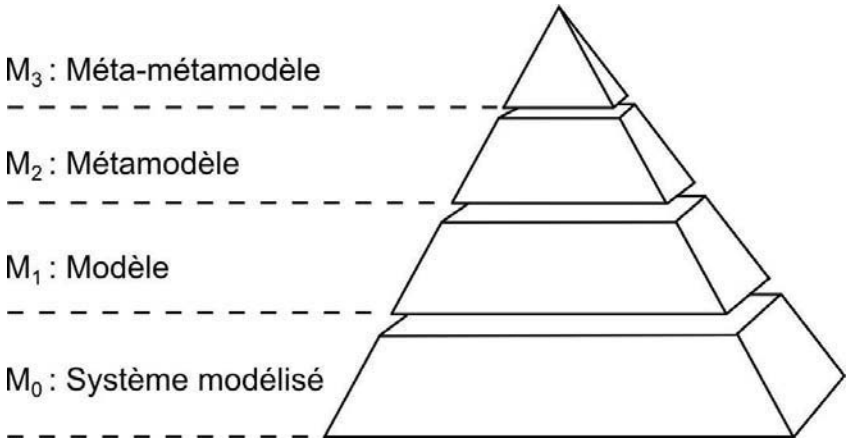


Figure 2.1.2 : La pyramide d'abstraction de l'IDM

La formalisation du métamodèle n'a pas été évoquée jusqu'à présent. Comme il s'agit d'un modèle (de langage), le formalisme adopté dépend de l'intention du modélisateur. Plusieurs possibilités sont offertes, différents espaces techniques pour reprendre l'appellation de (Kurtev *et al.* 2002). Un espace technique est défini par l'ensemble des outils et techniques issus d'une pyramide de métamodèles dont le sommet est occupé par une famille de méta-métamodèles similaires. La figure 2.1.3, issue de (Favre *et al.* 2006b), propose quatre exemples d'espaces techniques :

- l'espace des grammaires défini par les langages de description tels que l'Extended Backus-Naur Form ;
- l'espace des modèles inspirés de l'IDM - bien que théoriquement tous ces espaces puissent faire partie de l'IDM - défini par les méta-métamodèles tels que le MOF ou encore Ecore ;
- l'espace des documents basé sur les langages de marquage tels que l'eXtended Markup Language (XML) ;
- l'espace des bases de données reposant sur les langages de description de schémas. Il est à noter que le niveau M_3 de cette pyramide ne dispose pas du principe de métacircularité, les langages de description de schémas n'étant pas auto-descripteurs.

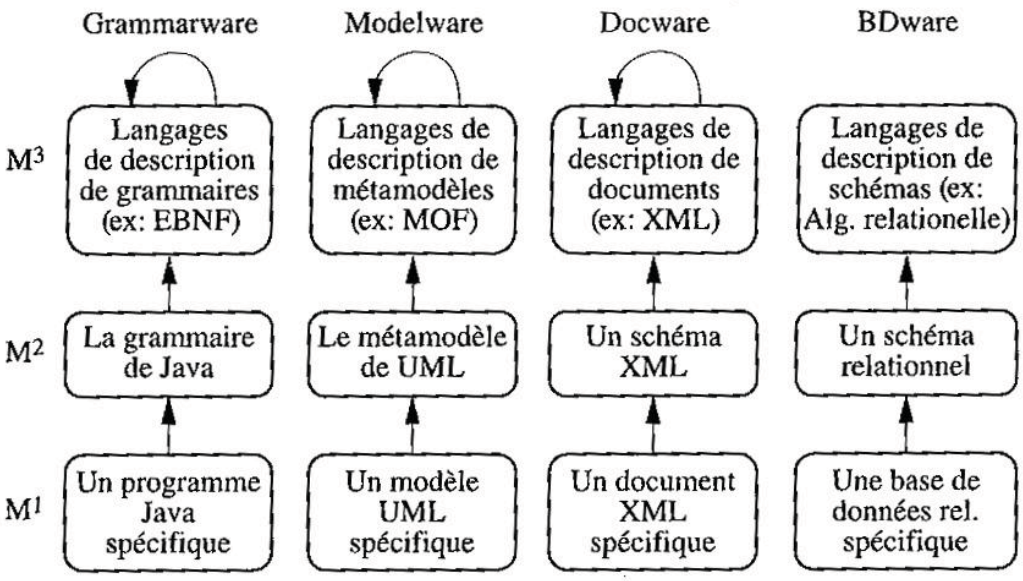


Figure 2.1.3 : Exemples d'espaces techniques proposés par (Favre *et al.* 2006b)

En définissant les mêmes concepts, au niveau M_2 de la pyramide, dans différents espaces techniques, il est possible de passer d'un espace technique à un autre en utilisant le concept de transformation.

1.1.3- Les transformations en IDM

C'est grâce à l'utilisation des transformations que l'IDM vise à rendre les modèles productifs (Selic 2003). En utilisant les concepts définis au niveau des métamodèles, les transformations permettent de passer d'un modèle à un autre. L'objectif des transformations est variable, ceci a conduit Tom Mens et Pieter Van Gorp à dresser une taxonomie des transformations dans (Mens et Van Gorp 2006). Ces auteurs distinguent les transformations endogènes des transformations exogènes.

Une transformation est qualifiée d'endogène lorsque le modèle cible et le modèle source sont conformes au même métamodèle. Ce type de transformation est utilisé pour enrichir le modèle source, ou encore pour la réingénierie du modèle (son *refactoring*). A l'inverse, une transformation exogène est appliquée lorsque les modèles source et cible sont conformes à deux métamodèles distincts, une telle transformation peut amener à changer d'espace technique. Ces transformations peuvent permettre une migration technologique ou encore la génération de code dans un langage de programmation généraliste. L'utilisation d'une succession de transformations est recommandée pour l'obtention de systèmes d'information complexes (Hemel *et al.* 2008).

Certains, notamment dans la communauté Eclipse, distinguent la génération de code, qualifiée de *Model To Text* (M2T), des autres transformations, désignées *Model To Model* (M2M). Pourtant, le code généré est bien un modèle (cf. Chapitre 1, Section 1) et il est possible de transformer le code en modèle par rétro-ingénierie (Favre et NGuyen 2004). La distinction entre les transformations M2T et M2M peut se comprendre puisque les espaces techniques, sur lesquels chacune renvoie, diffèrent. Cependant, elle résulte peut-être d'une vision encore trop centrée sur le code dont l'IDM cherche, avant tout, à s'affranchir. En effet, une fois le code obtenu, l'ingénieur logiciel peut considérer le travail abouti, alors que l'IDM ambitionne de couvrir également la maintenance et l'évolution des applicatifs, la notion de modèle devrait donc persister.

Dans cette présentation des concepts fondamentaux de l'IDM, l'omniprésence de la notion de langage est notable. Ceci rejoint notre question ouverte du Chapitre 1, l'IDM est axée sur la modélisation et la modélisation n'est-elle pas axée sur l'utilisation d'un langage ? (Mens et Van Gorp 2006) notent la similarité de concepts entre les transformations exogènes/ endogènes et les transformations telles que définies dans (Visser 2005). Ce dernier qualifie de reformulation (*rephrasing*) des transformations endogènes et de traduction (*translation*) des

transformations exogènes. La partie suivante vise à présenter les notions de Domain-Specific Language (DSL) et de Domain-Specific Modelling Language (DSML) mises en œuvre par l'IDM.

1.2- Les notions de langages dans l'IDM

Comme écrit ci-avant, une approche de métamodélisation conduit à l'identification d'un domaine restreint, spécifique, sur lequel le métamodèle va porter. Par cette restriction, il est alors possible de définir un langage de haut niveau d'abstraction. Ce langage spécifique à un domaine est appelé DSL (Domain-Specific Language). Sa spécificité offre une plus grande concision par rapport à un langage de programmation généraliste résultant ainsi en une amélioration de la productivité et une réduction des coûts de maintenance (Mernik *et al.* 2005). Le métamodèle joue un rôle central dans la définition du DSL, cependant il n'en constitue qu'une partie. (Gray *et al.* 2007) livre la définition formelle d'un langage *L* :

$L = <C, A, S, M_S, M_C>$
$M_S : A \rightarrow S$
$M_C : C \rightarrow A$

Le langage est constitué de la syntaxe abstraite *A*, celle-ci précise les concepts contenus dans le langage, leur articulation et règles d'association. Cette syntaxe n'est pas destinée à être manipulée directement par le modélisateur lors de l'utilisation de *L*. C'est le rôle de la syntaxe concrète *C*, elle peut être textuelle ou graphique. L'application M_C établit une correspondance entre les éléments de la syntaxe concrète et ceux de la syntaxe abstraite. *S* désigne la sémantique du langage, c'est-à-dire le sens des concepts portés par la syntaxe abstraite. Bien que la théorie définisse l'application M_S, qui assure la correspondance entre la syntaxe abstraite et la sémantique, cette dernière n'est pas systématiquement explicitée. Dans l'exemple donné par Gray et ses coauteurs, *L* est utilisé pour décrire un programme destiné à être exécuté. Dans ce cas, la sémantique ne trouve son sens qu'au moment où les expressions construites à l'aide de *C* sont exécutées. Il s'avère que la définition de la sémantique du langage reste une problématique d'actualité (Völter 2011; Selic 2012).

Gray et ses coauteurs (Gray *et al.* 2007) font la différence entre les DSLs et les DSMLs (Domain-Specific Modeling Languages). Les premiers ont une syntaxe concrète textuelle et les seconds une syntaxe concrète graphique. La terminologie domain-specific visual language (DSVL) semble indifféremment employée par d'autres auteurs pour désigner un DSML (Sprinkle et Karsai 2004; Mernik *et al.* 2005). Malgré l'intérêt que présenteraient des

solutions hybrides (textuelles et graphiques), celles-ci ne sont encore que peu employées (Völter 2011). Cela relève peut-être de la distinction faite entre les langages destinés à des informaticiens par rapport à ceux destinés à des utilisateurs non informaticiens. Les premiers gardant une vision trop centrée sur le code et trouvant ainsi plus pratique un langage textuel, alors que les seconds seront considérés plus à même de manipuler un langage visuel. Cette interprétation peut paraître forte, pour autant il est difficile de comprendre le manque d'intérêt manifeste pour les DSLs hybrides sans y faire appel.

La notion de DSL n'est pas nouvelle, le LOGO, le SQL ou l'HTML sont des exemples de DSLs textuels largement éprouvés. Quant aux langages visuels, Fabrik, proposé sur Macintosh grâce au Smalltalk (cf. figure 2.1.4), est un des langages les plus anciens dans le domaine (Ingalls *et al.* 1988).

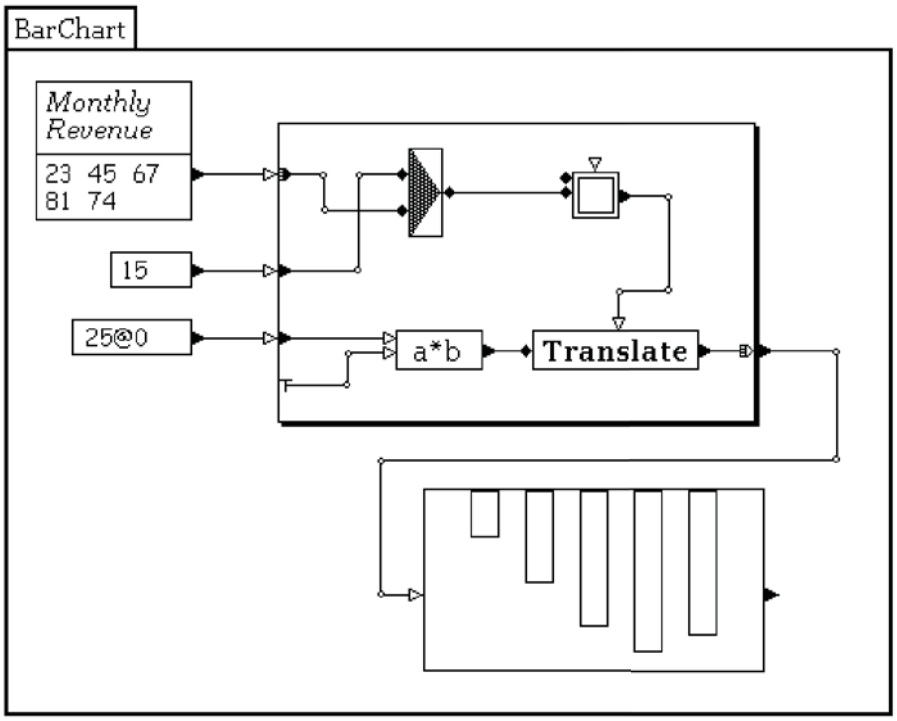


Figure 2.1.4 : Exemple de construction d'histogrammes par programmation visuelle à l'aide de Fabrik (source : (Ingalls *et al.* 1988))

Une bibliographie étendue sur les DSLs était déjà publiée par (vanDeursen *et al.* 2000). La terminologie « *model-driven* » en est absente, tout comme les concepts de l'IDM, ils sont cependant sous-jacents notamment dans la démarche de l'ingénierie de domaines (Arango 1989).

Fournir un langage de haut niveau d'abstraction n'est pas chose aisée, tant par le fait que les abstractions doivent satisfaire correctement les besoins du domaine d'application, que par la multiplicité des techniques qui peuvent intervenir dans l'obtention du DSL. Nous allons maintenant aborder les étapes essentielles d'une démarche IDM en précisant certaines solutions techniques qui permettent d'accomplir ces étapes.

1.3- Mises en œuvre de l'IDM

La mise en œuvre d'une démarche IDM n'est pas aisée, nombreuses sont les publications abordant le sujet et listant les écueils à éviter, par exemple (Mernik *et al.* 2005; Kelly et Tolvanen 2008; Kelly et Pohjonen 2009; Völter 2011). Toutes s'accordent sur la première étape à aborder, il s'agit de l'appréhension du domaine. Déjà évoquée par (vanDeursen *et al.* 2000) pour la construction d'un DSL, elle reste indispensable dans le cadre d'une démarche IDM (Kelly et Pohjonen 2009; Cuadrado *et al.* 2012). Différentes modalités sont accessibles pour la découverte du domaine, dans tous les cas, sous la forme d'un processus itératif (VanDeursen *et al.* 2007; Kelly et Tolvanen 2008). (Cuadrado *et al.* 2012) propose un processus faisant intervenir les experts du domaine. Ceux-ci réalisent une modélisation informelle à partir de laquelle les concepts du domaine sont identifiés ainsi que la syntaxe concrète du DSML. Plus classiquement, la démarche peut débuter par l'exploitation du patrimoine applicatif (le *legacy*). (Rugaber et Stirewalt 2004), ainsi que (Favre 2004b), ont introduit le concept de *Model-Driven Reverse Engineering*[9] qui permet à partir de codes existants de retrouver les modèles qui les décrivent. Ce concept fait appel aux techniques de rétro-ingénierie et de réingénierie (Favre *et al.* 2006a). Encore une fois, la rétro-ingénierie et la réingénierie ne sont pas nouvelles et existaient bien avant l'IDM (Chikofsky et Cross 1990). La manière dont la rétro-ingénierie est mise en œuvre est conditionnée par le code existant, il peut s'agir d'un unique système informatique complexe ou alors d'un ensemble de systèmes informatiques appartenant au même domaine.

[9] Rétro-ingénierie dirigée par les modèles

La figure 2.1.5 est une représentation d'une démarche IDM menée à partir d'un ensemble de systèmes informatiques d'un même domaine. Cette représentation est donnée à titre d'exemple, il n'existe pas à notre connaissance de démarche standardisée. Plusieurs systèmes informatiques[10] (S_1.. S_n) existants sont soumis à une étape de rétro-ingénierie (1) aboutissant à l'obtention de leurs modèles respectifs (M_1..M_n). Ces modèles fournissent une vision synthétique des différents applicatifs. Grâce à eux, il devient plus facile d'identifier les concepts clés du domaine et leurs règles d'association et de concevoir ainsi un méta-modèle du domaine (2). Cette étape peut se réaliser en parallèle de la réingénierie des systèmes informatiques (S'_1.. S'_n) (3). En effet, la définition d'un cadriciel commun (4) permet de mieux cerner la sémantique associée au métamodèle et de manipuler les mêmes concepts. La

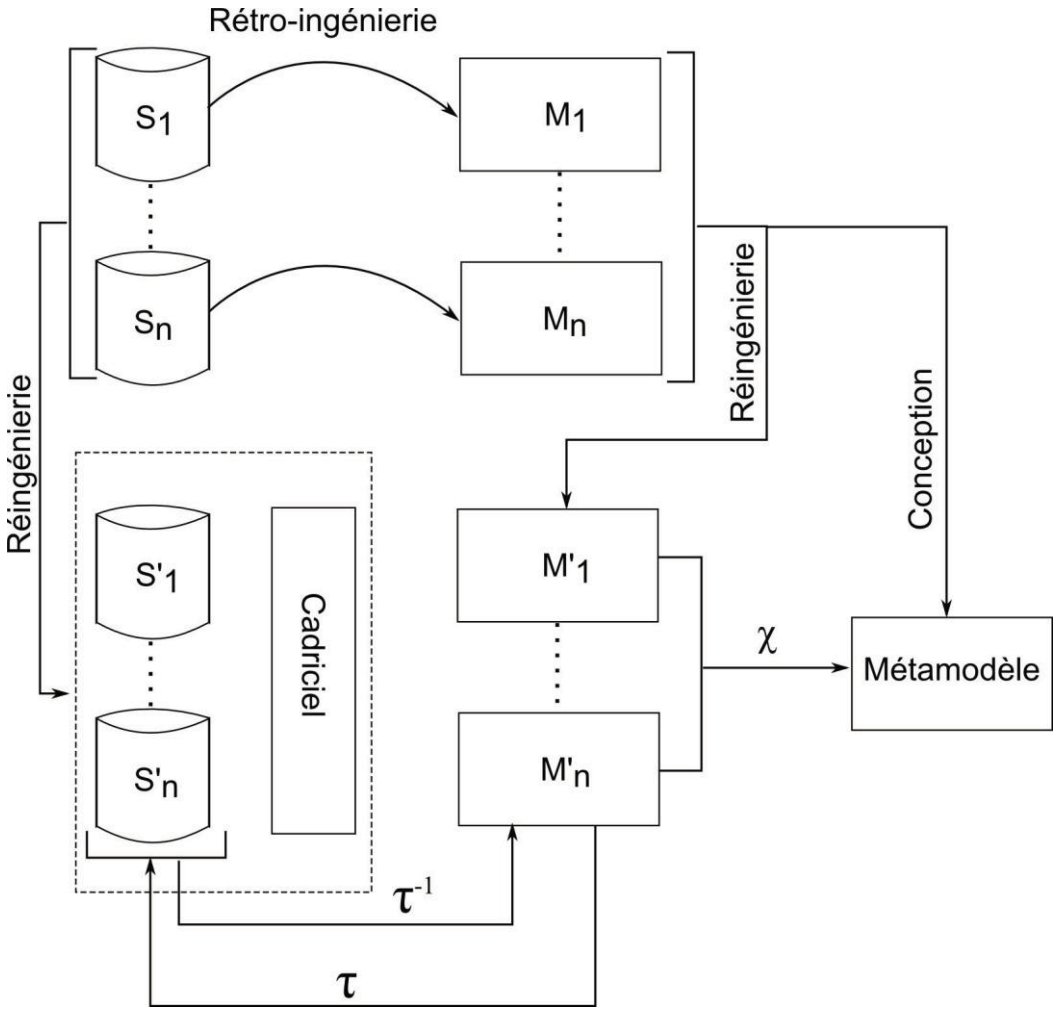


Figure 2.1.5 : Approche dirigée par les modèles à partir d'un ensemble de systèmes informatiques patrimoniaux appartenant à un même domaine

[10] Nous entendons ici, le code sur lequel repose le système informatique

spécification de la syntaxe concrète complète le DSML (5), il devient alors possible de procéder à la réingénierie des modèles (6). Dans le cas de figure retenu, il existe une correspondance parfaite entre le modèle conforme au métamodèle et le système informatique s'appuyant sur le cadriciel, dès lors la transformation du modèle en code τ (7) est bijective, sa réciproque τ^{-1} permet donc à partir du même code de retrouver exactement le même modèle.

Plusieurs éléments sont à noter concernant cette démarche. En premier lieu, il est peu probable qu'elle soit conduite de manière rectiligne, c'est-à-dire qu'elle mène à des propositions successives de différents éléments : métamodèle, cadriciel et syntaxe concrète connaîtront plusieurs versions successives. Le DSML sera tout d'abord réalisé sous la forme d'un prototype (Kelly et Pohjonen 2009), ceci afin d'en limiter les coûts de production. De plus, suivant la nature des systèmes informatiques considérés, certaines étapes peuvent n'être que partielles. En effet, si leur comportement est facilement identifiable et testable, il peut être intéressant de comparer un système S_i avec la nouvelle implémentation S'_i obtenue à partir de la transformation de M'_i. Ceci permet d'éviter la réingénierie de tous les systèmes et de tester la validité de la démarche. Comme pour toute démarche de production, procéder à des tests est essentiel à la fois dans le cadre de la conception du DSML, du cadriciel (*framework*) et des transformations (DenHaan 2008). Cependant, ces tests doivent aussi être employés une fois que ces derniers ont atteint une forme mature. Cette activité est couverte par la vérification et la validation (Merilinna et Pärssinen 2010). L'article de Merilinna et Pärssinen propose une revue des moyens disponibles pour la vérification des métamodèles, des modèles construits à partir d'un DSML ainsi que du code généré et la validation des codes obtenus vis-à-vis des spécifications d'origine. Il n'existe pas d'outils intégrés permettant de procéder à la vérification et la validation des environnements de conception basés sur des DSML et des modèles qu'ils permettent de construire. Des démarches intégratives à base de modèles ont été proposées (Merilinna *et al.* 2008) sans pour autant permettre de couvrir l'ensemble des besoins.

L'une des tâches les plus délicates est la validation du code généré par rapport aux spécifications. Comme nous l'évoquions auparavant, la sémantique du DSML est souvent implicite, elle se concrétise par l'exécution du code généré. Dans ce cas, la validation de l'application requiert une comparaison avec la spécification d'origine ou encore la génération automatique de tests, une revue sur cette thématique est offerte par (Mussa *et al.* 2009). (Kelly et Tolvanen 2008) propose également une construction progressive des générateurs de code et

l'utilisation d'applications types pour procéder à la validation du code généré. Une fois cette validation effectuée, les auteurs argumentent que la stabilité du générateur de code et sa construction permettent de penser que tout nouveau code généré sera correct. Ce schéma de pensée n'est cependant pas acceptable dans le cadre de la production d'applications critiques (Merilinna et Pärssinen 2010). Par contre, lorsque le DSML est exécutable, le comportement spécifié grâce au modèle est évalué dans l'environnement de conception et peut permettre la génération automatique de scénarios de validation du code généré comme dans (Conrad 2009).

La diversité de conduite des opérations de vérification et de validation trouve son origine dans la qualité même de l'IDM : sa spécificité. En effet, pour le cas que nous venons d'évoquer, un DSML n'a pas obligatoirement vocation à être exécuté. Son rôle est conditionné par l'intention des spécialistes du domaine qui peuvent, ou non, requérir l'exécutabilité des modèles qu'ils conçoivent. La nature même des systèmes cibles peut avoir un impact sur les opérations de vérification et de validation qui peuvent être menées. Dès lors, il s'avère difficile de proposer des outils de vérification et de validation qui puissent être appliqués à n'importe quel DSML.

Il en va de même pour la définition de l'approche à adopter dans le cadre d'une démarche IDM. La figure 2.1.5 place cette démarche dans le cadre d'un ensemble de systèmes informatiques ayant la même teneur et dont le domaine peut être entièrement décrit dans un unique métamodèle. Le cas introduit par la figure 2.1.6 est différent, elle considère que le *legacy* est un unique système d'information d'une taille importante, (Favre *et al.* 2006a) évoque le cas de Dassault Systèmes dont la partie implémentation du système d'information[11] regroupait plus de 8 millions de ligne de code. Bien que coûteuse, une approche IDM et de rétro-ingénierie a un apport non négligeable dans un tel cas. Elle offre la possibilité de formaliser le patrimoine de la société et, une fois menée à bien, de faciliter les futures évolutions des systèmes informatiques ou leur migration technologique.

Dans ce cas précis, construire un unique modèle pour ce système informatique, à l'instar de l'opération illustrée précédemment par la figure 2.1.5, n'aurait que peu de sens du point de vue de sa compréhension, rappelons ici le caractère essentiel que revêt la simplification dans

[11] Un système d'information (SI) est un ensemble organisé de ressources (matériels, logiciels, personnels, données et procédures) qui permet de regrouper, de classifier, de traiter et de diffuser de l'information. Notons que celui-ci ne se limite pas aux ressources informatiques.

une opération de modélisation (cf. chapitre 1). Selon Favre et ses coauteurs, deux voies de simplification s'offrent alors :

- l'extrême simplification du système informatique afin de procéder à la redécouverte de l'architecture. Elle permet de construire un modèle unique qui permettra de mieux conduire la deuxième opération ;
- l'identification de différents sous-ensembles du système informatique pour lequel il est possible de mener une opération de rétro-ingénierie détaillée.

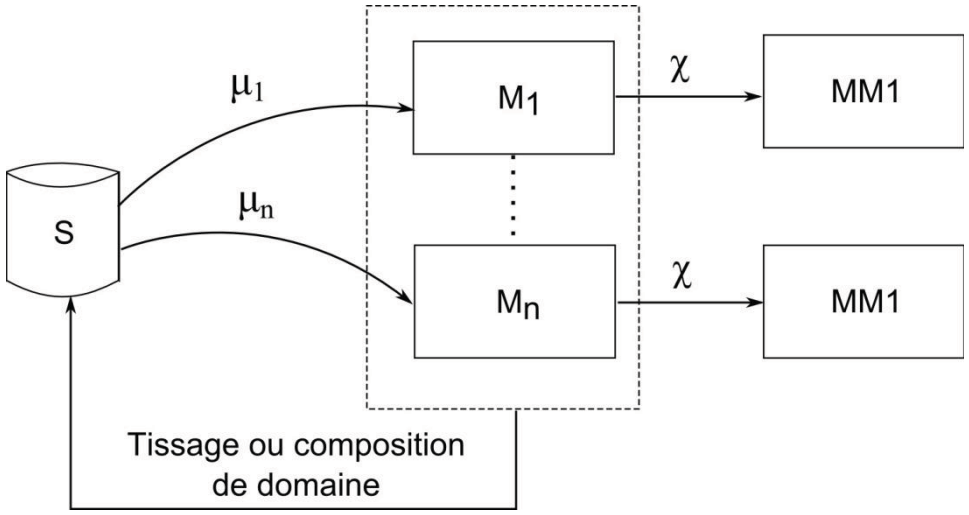


Figure 2.1.6 : Approche de métamodélisation d'un système informatique complexe

Cette dernière voie est représentée dans la figure 2.1.6, les différents modèles sont des sous-ensembles du système informatique focalisés sur une activité précise de celui-ci. Chacun des modèles offre donc un point de vue sur un domaine précis du système informatique, il est donc possible de définir le méta-modèle correspondant à ce domaine.

Notons ici le changement de relation entre les différents modèles et le système informatique d'origine, chaque modèle M_i est une représentation simplifiée du système informatique ce qui justifie l'emploie de la relation μ_i. Il n'est pas possible à partir de M_i d'obtenir par transformation l'ensemble du système informatique. L'ensemble des modèles $M_1..M_n$ est nécessaire pour cela. Comme le souligne (Bézivin *et al.* 2005), cette opération ne peut reposer sur une simple additivité des modèles. Elle relève du domaine de l'AOM (*Aspect Oriented Modeling*). Plusieurs pistes sont possibles pour parvenir au tissage des différents modèles afin d'obtenir le code définitif par la composition de DSL (Estublier *et al.* 2005) ou encore l'utilisation de transformations de graphes (Whittle et Jayaraman 2007).

Que ce soit en termes de démarche possible ou de solutions techniques pour une activité, l'IDM offre un large panel de choix. Comme nous le soulignions précédemment, cela est sans doute lié à la spécificité des domaines qui peut faire naître des besoins très différents d'un cas d'utilisation à un autre. Il est également vraisemblable que la relative jeunesse de l'IDM en soit responsable et qu'à terme une homogénéisation se mette en place avec une plus large adoption du paradigme « tout est modèle ». Une mise en regard des éléments présentés dans cette partie avec la problématique de la thèse définie au Chapitre 1 nous permet de poser de sérieux arguments justifiant la mise en œuvre d'une démarche IDM pour répondre à cette problématique. Tout d'abord les DSMLs sont des langages de haut niveau d'abstraction permettant à des experts - pour notre cas les modélisateurs en agronomie - de manipuler les concepts de leur métier et de définir des modèles. A partir de ces derniers, il est possible d'obtenir le code correspondant au modèle spécifié à l'aide de transformations et donc, dans notre cas, d'obtenir le code Java du modèle agronomique. De plus, il est possible de rendre le DSML exécutable et donc de permettre, à partir de l'environnement de conception, la réalisation de simulations ce qui pourrait affranchir le modélisateur de la phase de prototypage Matlab. Ce qui *in fine* permettrait la définition de scenarios de test pour procéder à la vérification et la validation du code généré. De plus, ITK dispose d'un patrimoine de plusieurs modèles agronomiques rendant possible d'envisager une démarche IDM du type de celle décrite par la figure 2.1.5. Avant de nous lancer dans une telle démarche, nous avons souhaité tester les potentiels de plusieurs solutions de modélisation et simulation généralistes (telles que Matlab), spécialisées au domaine de l'agriculture ou encore issues de démarches IDM.

2- Les outils existants pouvant répondre aux problématiques de la thèse

2.1- Les outils de modélisation classique

2.1.1- Matlab/Simulink

Matlab® (Mathworks^{TM}) est un outil dédié au développement de modèles scientifiques. Il est reconnu dans beaucoup de domaines, une simple recherche sur ScienceDirect des articles ou ouvrage référençant Matlab dans leur titre ou leurs mots-clés renvoie 4000 références. En premier lieu, il offre un langage procédural ; depuis sa version 2008, il est également possible de développer en objet.

Parmi les caractéristiques les plus importantes de Matlab, il faut noter son intégration native du calcul matriciel. La manipulation de variables vecteurs et matrices facilite l'écriture des algorithmes. Dans le cadre de Matlab, l'utilisation des matrices offre des performances de calcul bien meilleures par rapport à l'utilisation d'une boucle pour réaliser un traitement équivalent. Outre la manipulation directe de matrices, l'une des plus-values de Matlab est son environnement de développement, qui permet :

- comme pour un IDE classique de débugger le code avec la possibilité d'interroger les variables en ligne de commande ;
- d'effectuer un contrôle des différentes données à la fin d'une simulation, en effet celles-ci peuvent être conservées en mémoire et directement manipulées par des lignes de commande ;
- de procéder à un tracé de courbes aisé, avec possibilité de faire des ajustements de tracé voire d'ajouter de nouvelles courbes après la simulation ;
- d'utiliser de nombreuses bibliothèques de calcul complémentaires (certaines nécessitent une licence supplémentaire).

Cet ensemble de fonctionnalités fait de Matlab un environnement très apprécié du modélisateur. Le fait que son langage soit interprété a des conséquences sur ses performances en termes de temps de calcul. Des fonctionnalités de génération de code compilé existent, soit en C ou C++, soit de génération de code Java avec le Matlab Builder JA. Cependant, le code produit par ce dernier est spécifique à une plate-forme ce qui rend caduque l'un des avantages reconnus de Java : sa portabilité.

Matlab propose également un environnement de conception visuelle, Simulink, celui-ci s'appuie sur une formalisation en diagramme de blocs. Cet environnement a été utilisé avec succès pour la construction de modèles de spécification en robotique (Shenoy *et al.* 2007) et présente un intérêt pour les domaines de la mécanique ou encore de la microélectronique. Cependant, son caractère multi-domaine a une contrepartie. En effet, il offre une profusion de formalismes qui peuvent dérouter le spécialiste d'un domaine pour lequel aucune bibliothèque de modélisation n'a été définie. C'est le cas de la modélisation « grande culture ». De plus, les formalismes proposés ne semblent pas permettre de gérer la modélisation de systèmes dynamiques à structure dynamique. Un outillage spécifique à la biologie a été mis en place avec SimBiology® mais il ne s'adresse qu'au domaine de la pharmacocinétique et de la pharmacodynamique dont les formalismes sont très éloignés de ceux employés en modélisation des cultures.

2.1.2- DEVS/VLE

Il serait difficile de proposer une étude sur le domaine de la modélisation et de la simulation sans aborder DEVS (Discrete EVent systems Specification). Proposé par Bernard P. Zeigler en 1976 (Zeigler 1976), DEVS est un formalisme de modélisation à événements discrets et également, puisqu'elle en est un sous-ensemble (Zeigler *et al.* 2000), de modélisation en temps discret. La formalisation originelle de DEVS est désormais désignée comme CDEVS (Classical DEVS), en effet, de nombreuses déclinaisons ou extensions de DEVS ont vu le jour : par exemple PDEVS (Chow et Zeigler 1994) pour la simulation en parallèle ou encore DSDEVS (Barros 1995) pour la modélisation de systèmes dynamiques à structure dynamique.

Nous ne pouvons faire une présentation complète de DEVS et de ses diverses implémentations dans le cadre de ce mémoire mais ne pouvons éviter quelques mots sur le formalisme de base en DEVS. Celui-ci repose sur la notion de modèle atomique. Un modèle atomique *M* est défini par le n-uplet suivant :

$$M = <X, Y, S, t_a, \delta_{int}, \delta_{ext}, \lambda>$$

Avec :

- X correspondant aux entrées du modèle, ce sont des informations issues d'événements extérieurs ;
- Y correspondant aux informations que M peut envoyer vers l'extérieur suite à un événement interne à M ;
- S correspondant à l'ensemble des états internes que M peut adopter ;
- t_a correspondant à une fonction qui à un état de S associe un temps caractéristique. Si ce temps caractéristique s'est écoulé sans perturbation extérieure, M change d'état ;
- ce changement d'état est déterminé par la fonction de transition interne de M : δ_{int}. Celle-ci permet de déterminer le nouvel état de M et de définir les informations envoyées vers l'extérieur ;
- l'arrivée d'informations depuis l'extérieur déclenche l'évaluation d'une fonction de transition externe qui, suivant les informations obtenues et l'état actuel du modèle, peut amener à changer l'état de M.

DEVS est conçu pour définir des modèles complexes et permettre leur hiérarchisation. Cette dernière s'effectue par l'emploi de modèles couplés. Un modèle couplé permet de contenir d'autres modèles couplés ainsi que des modèles atomiques. L'un des points forts de DEVS est qu'il a été prouvé mathématiquement que les modèles couplés se comportent de la même manière que les modèles atomiques. Ceci, ainsi que la preuve mathématique de la modalité d'exécution du modèle, permet de donner un cadre formel aux modèles implémentés grâce à DEVS. Un autre point fort de DEVS est de permettre la construction de modèles hétérogènes, c'est-à-dire employant des modèles reposant sur différents formalismes. Cela étant, et comme le souligne Luc Touraille dans sa remarquable thèse (Touraille 2012), DEVS ne doit pas être vu comme le formalisme de modélisation ultime mais plutôt comme un formalisme trans-standard permettant de faciliter l'intégration et la simulation de modèles hétérogènes.

Parmi les travaux publiés autour de DEVS, notre attention a été retenue par VLE (*Virtual Laboratory Evironment*) (Quesnel *et al.* 2009). Cet environnement mis au point à l'Université du Littoral Côte d'Opale permet de gérer l'ensemble du cycle de modélisation et de simulation de la conception à l'analyse des résultats. Il présente l'avantage d'intégrer une extension DSDEVS essentielle dans le cadre de la modélisation en agronomie, un environnement de conception visuelle grâce à GVLE (*Graphical VLE*) et d'avoir été utilisé pour l'implémentation d'un modèle FSPM : ecomeristem (Quesnel *et al.* 2012). Cependant,

outre les difficultés d'installation rencontrées lors de son évaluation, l'interface graphique fournie par GVLE devient difficile à manipuler lorsque le modèle conçu gagne en complexité (Bergez *et al.* 2013). De plus, il n'existe pas d'extension DEVS dans VLE permettant de faciliter la spécification de modèles à temps discret. Enfin, cet environnement ne nous a pas paru adapté à une manipulation directe par des modélisateurs d'ITK. En effet, son utilisation requiert encore d'importantes compétences informatiques (Soulié 2011).

2.2- Les outils de modélisation spécialisés pour l'agronomie

Les outils de modélisation spécialisés pour l'agronomie sont, à notre connaissance, exclusivement issus du monde de la recherche. Bien que reposant sur des technologies différentes, ils semblent tous avoir une motivation commune : la fédération de la communauté de modélisateurs agronomes. Cette motivation commune a amené à une convergence des solutions techniques par la mise en œuvre d'une approche dite par composants. Dans ce contexte, les plateformes ou environnements de modélisation sont assez nombreux, nous avons retenu, dans le cadre de cet état de l'art, ceux qui nous sont apparus comme étant les plus proches de nos besoins.

2.2.1- La plateforme DSSAT

DSSAT (Decision Support System for Agrotechnology Transfer) (Jones *et al.* 2003) est l'une des productions majeures du projet IBSNAT (International Benchmark Sites Network for Agrotechnology Transfer). Au départ, ce système a été conçu et développé par des agronomes pour évaluer le résultat et les risques de différents modes de gestion de cultures. Par la suite, l'intégration de nouvelles technologies informatiques a nécessité l'intervention d'une équipe d'informaticiens aguerris. Son développement s'est ensuite poursuivi au sein du consortium ICASA (International Consortium for Agricultural Systems Applications) qui a pris fin en 2011. La poursuite du développement est désormais assurée par le réseau DSSAT (www.dssat.net). DSSAT est conçu comme un outil d'analyse et de simulation pour une vingtaine de cultures différentes. Les fonctionnalités offertes ne se limitent pas à l'utilisation d'un modèle grande culture. Elles permettent également de modéliser les ravageurs des cultures, de tester différentes pratiques culturales, d'effectuer des évaluations de modèles comme des analyses de sensibilité et de gérer les bases de données qui leur sont associées (cf. figure 2.2.1)

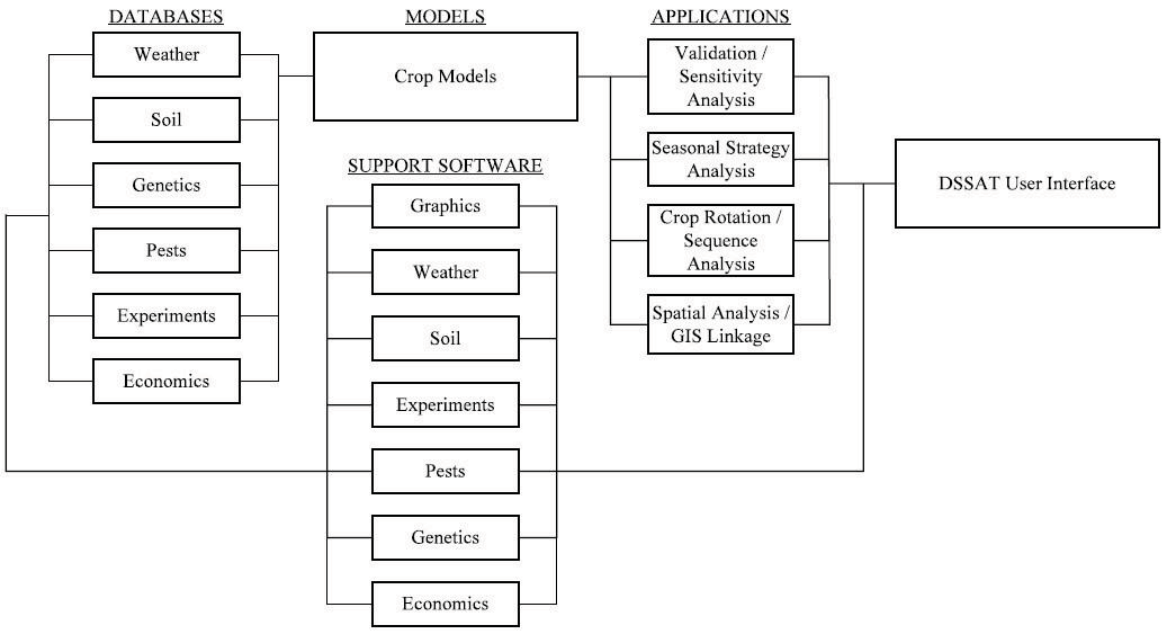


Figure 2.2.1 : Description des différents composants de DSSAT tel que publiée dans (Jones *et al.* 2003)

Cet outil très complet a pour vocation d'aider les chercheurs, exploitants ou conseillers techniques du domaine de l'agriculture dans leur processus de décision ou d'évaluation. Il n'est pas, à proprement parler, un outil de conception de nouveaux modèles. En effet, il propose d'exploiter des modules préexistants. Il est théoriquement possible d'intégrer de nouveaux formalismes dans cette plateforme, nous n'avons cependant pas pu tester cette fonctionnalité. La dernière version de DSSAT semble en effet poser quelques problèmes lorsqu'elle est utilisée sur les dernières versions des systèmes d'exploitation les plus courants. Il est fort probable que la prochaine version de DSSAT dont la sortie est prévue pour 2013 permette de résoudre ces problèmes de stabilité.

2.2.2-La plateforme RECORD

Le projet RECORD[12] a été initié en 2005 ; porté par les départements Environnement et Agronomie (EA) et Mathématique et Informatique Appliquées (MIA) de l'Institut National de la Recherche Agronomique (INRA), son objectif est de mettre en place une plateforme de modélisation multidisciplinaire afin de concevoir, d'évaluer et d'analyser des systèmes de culture en prenant en compte : les cultures, les techniques agricoles, les conditions économiques, les bioagresseurs (maladies et insectes). Originellement orienté vers les

[12] RECORD : REnovation et COoRDination de la modélisation des cultures pour la gestion des agro-écosystèmes

modélisateurs des départements EA et MIA, le projet a finalement été conçu comme un outil devant répondre aux besoins de l'ensemble des modélisateurs de l'INRA et de certains de ses partenaires. RECORD devait permettre une approche visuelle de couplage de modules ainsi que l'utilisation de formalismes hétérogènes, tout en étant un environnement intégré de M&S sous une licence logiciel libre.

Après étude de différents environnements existants, le projet s'est orienté sur l'utilisation de VLE comme environnement de base et de l'améliorer par des constructions spécifiques (Bergez *et al.* 2013). Parmi celles-ci, des formalismes ont été intégrés afin d'éviter aux modélisateurs l'emploi direct du formalisme DEVS. Ces formalismes, définis sous forme d'extensions, regroupent les équations aux différences, les équations différentielles ordinaires ainsi que la définition de plans de gestion des cultures. Ces extensions sont implémentées en C++. Pour faciliter l'utilisation par les modélisateurs, des fenêtres de saisie d'équations ont été développées. Une fois la saisie effectuée, le code C++ correspondant est généré. Il n'existe pas de fonctionnalité équivalente permettant de définir un algorithme complet pour la représentation d'un processus biophysique donné. Pour ce faire, la seule possibilité est d'éditer directement le code C++ du modèle, solution qui ne peut être retenue dans le contexte de notre étude. Selon Bergez et ses coauteurs, l'interface graphique, qui s'appuie sur GVLE, est surtout utile quand le nombre de modèles reste faible. En effet, un modèle du même ordre de complexité que ceux utilisés chez ITK pose des problèmes de lisibilité (cf. Figure 2.2.2).

RECORD est une initiative ambitieuse de l'INRA, elle a amené à la reprise de modèles existants tels que STICS (Brisson *et al.* 1998) et à leur adaptation pour le formalisme à événement discret. La plateforme cible un panel large d'activités de modélisation des systèmes de culture. Elle est donc contrainte à offrir un formalisme peu spécifique ce qui en limite son expressivité. Dans la figure 2.2.2, n'apparaît pas la notion de séquence d'exécution, ceci est inhérent à la gestion événementielle de DEVS. Dans ces conditions, il peut être délicat pour le modélisateur de se repérer et même d'appréhender la logique d'exécution de son modèle. Cette plateforme n'est pas dotée de fonction de génération de code permettant de passer à un autre environnement, l'utilisation de JVLE pourrait permettre d'exécuter les modèles conçus à partir de RECORD sur une plateforme Java. Enfin, se soulève la problématique de la licence et de droit d'exploitation commerciale de modèles qui pourraient être développés dans RECORD. La charte qui est en cours de négociation pour les utilisateurs de cette plateforme prévoit dans sa version intermédiaire que tous les modèles développés à

partir de RECORD sont accessibles à l'ensemble de sa communauté. Une telle clause poserait des problèmes de compatibilité avec l'activité commerciale d'ITK.

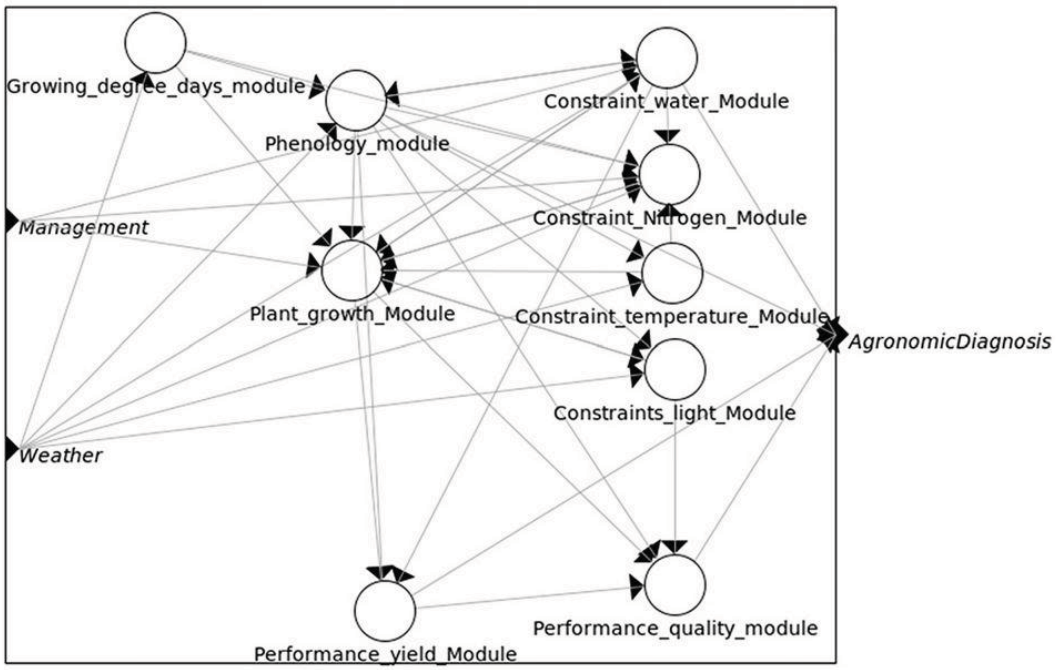


Figure 2.2.2 : Le modèle biophysique de SUNFLO dans l'interface de conception de RECORD tel que publié dans (Bergez *et al.* 2013)

2.2.3- OpenAlea

OpenAlea (Pradal *et al.* 2008) est un outil sous licence LGPL[13] dont l'objectif premier est de fournir un environnement de M&S permettant le partage de représentations dans la communauté de chercheurs sur la modélisation architecturale des plantes à différentes échelles spatiales et temporelles. OpenAlea a été développé en Python. Comme pour les autres environnements de M&S en agronomie développés par les acteurs de la recherche, son approche fédératrice, permettant la réutilisation de modèles, s'est naturellement orientée sur une architecture à base de composants. OpenAlea permet la réutilisation de modèles implémentés dans d'autres langages tels que le C, le C++ ou le Fortran.

OpenAlea est doté d'une interface graphique de conception de modèles : VisuAlea. Cette interface propose de manipuler des boîtes (les nœuds) et des lignes (les flux de données) pour définir un dataflow. Les boîtes permettent d'adopter différents niveaux de granularité. En effet, elles permettent de définir des nœuds composites contenant eux-mêmes un dataflow. Un

[13] Lesser General Public Licence

nœud atomique peut contenir une fonction, certains nœuds prédéfinis (p.ex. le nœud '+' qui additionne deux valeurs) permettent de construire des diagrammes de bloc. Cette formalisation s'éloigne des choix de représentations naturellement adoptés par les modélisateurs d'ITK pour la modélisation conceptuelle (cf. Figure 2.2.3).

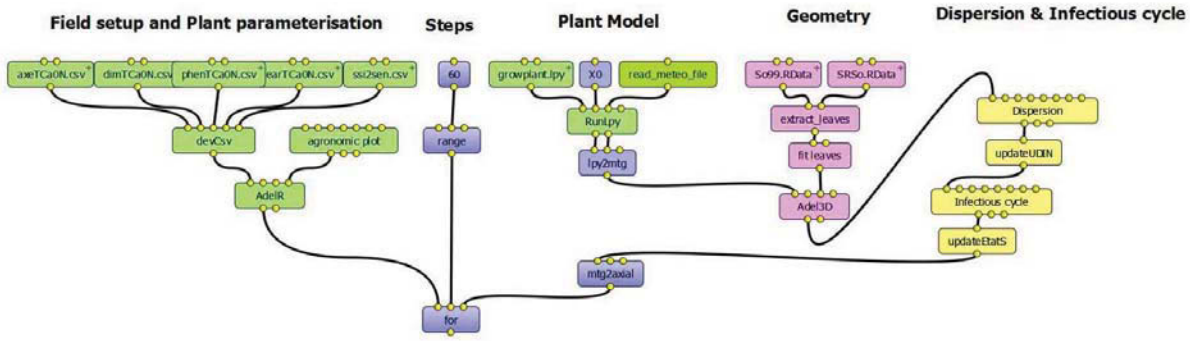


Figure 2.2.3 : Modèle d'une interaction plante-pathogène à l'aide de VisuAlea tel que publié dans (Fournier *et al.* 2010)

Dans VisuAlea, la définition du dataflow est stricte, c'est-à-dire qu'il est impossible d'avoir une circularité dans la transmission des informations entre les différents nœuds (cf. figure 2.2.4 d). En effet, pour pouvoir exécuter la simulation, l'environnement détermine l'ordre d'exécution des différents nœuds en fonction des informations qu'ils nécessitent : dans la partie (a) de la figure *A* est exécuté puis *B* et enfin *C*. Une circularité dans le dataflow empêcherait cette détermination.

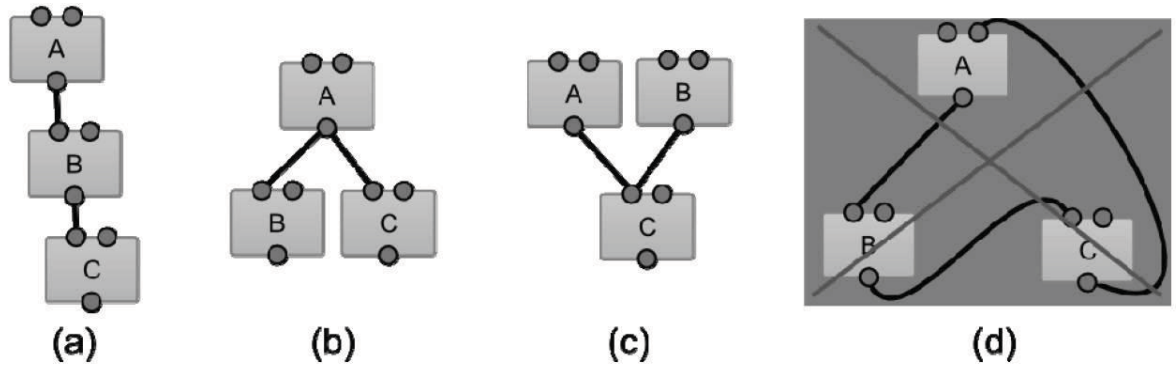


Figure 2.2.4 : Exemples de dataflow conçus à partir de VisuAlea (a, b, c) et d'une construction interdite (d) tels que publiés dans (Chopard *et al.* 2011)

Certains choix dans la conception du dataflow peuvent paraître surprenants. L'information transmise par la sortie d'un nœud (partie b de la figure) est transmise par référence aux nœuds qui l'utilisent en entrée. Les nœuds *B* ou *C* peuvent modifier cette valeur or leur ordre

d'exécution n'est pas garanti (Chopard *et al.* 2011), cela peut donc fausser le résultat de la simulation. La liaison de plusieurs sorties a une seule et même entrée (partie c de la figure) est possible. Dans ce cas, l'entrée reçoit une liste de valeurs dont l'ordre dépend de la disposition des nœuds sur le diagramme, le plus à gauche fournira la première valeur de la liste. Si l'ordre des valeurs n'a pas d'importance pour le traitement de *C* ceci ne posera pas de problème. Par contre si cet ordre est important, ce choix d'implémentation peut avoir un impact sur la réutilisabilité des composants.

Le formalisme adopté dans VisuAlea ne répond pas aux besoins exprimés au chapitre 1. De plus, OpenAlea ne nous semble pas doté de fonctionnalités de génération de code notamment pour la plateforme Java. Enfin, le partage de modèles entre les différents chercheurs, s'il est souhaité et effectif, manque encore de formalisation en termes de propriété intellectuelle ; ceci fait l'objet de discussions au sein de la communauté OpenAlea.

2.3- Les outils associés à l'IDM

2.3.1- Modelica

Modelica est un langage dédié à la modélisation multi-domaine, en permettant d'intégrer dans un même modèle des représentations de systèmes mécaniques, hydrauliques ou encore électriques. Il a été mis en place à la fin des années 90 par une action conjointe du monde de la recherche et l'industrie (Mattson et Elmqvist 1997), ces acteurs se sont regroupés sous la forme d'une association : l'association Modelica[14]. Cette association se focalise sur la définition et la promotion du langage, non sur la conception d'environnements de modélisation et de simulation qui l'exploitent. De nombreuses implémentations à but commercial ou issues du monde du logiciel libre existent parmi lesquelles : le WolframSystemModeler™ de Wolfram Inc., MapleSim™ de MapleSoft ou encore OpenModelica® développé par l'université de Linköping et l'Open Source Modelica Consortium.

La considération de base qui a orienté la conception de Modelica est qu'un système que l'on cherche à modéliser est décomposable en sous-systèmes (Fritzson 2010a) dont le comportement individuel est modélisable. Il s'agit d'une approche cartésienne classique et d'une vision orientée système physique (au sens de la discipline du même nom), pour laquelle un sous-système est régi par une équation ou un jeu d'équations. Par exemple, la loi d'Ohm

[14] https://www.modelica.org/

décrit par une équation simple le lien entre tension, intensité et résistance aux bornes d'un dipôle électrique U= R* I. Cette équation, formalisée dans une classe, peut être réutilisée en plusieurs endroits d'un même circuit électrique. Tous les composants électriques régis par la loi d'Ohm seront des instances de cette classe ou d'une de ses classes filles. Ce système de représentation permet de procéder à de la modélisation 'non causale' qui est au cœur de Modelica. En conservant l'exemple de la loi d'Ohm, cela signifie qu'au besoin la tension pourra servir d'entrée pour déterminer l'intensité ou inversement. Ceci a conduit à l'introduction de la terminologie de langage de programmation orienté objet à base d'équations (*object-oriented equation-based programming language*).

Nous présentons Modelica dans la section réservée aux outils associés à l'IDM, bien qu'à l'origine, cet outil ne résulte pas d'une démarche IDM. Cependant plusieurs éléments nous rapprochent de la vision d'espace technique telle que définie en première partie de ce chapitre. Notamment, la syntaxe concrète de Modelica est spécifiée à l'aide de l'extended Bakus-Naur Form. Elle définit un langage spécifique au domaine - un DSL - de la modélisation. Les différentes équations définies pour un système permettent la déduction d'un ensemble de relations par exploration logique (Fritzson et Engelson 1998). Cette exploration est rendue possible par la sémantique associée à Modelica et renvoie à la notion de transformation. Tout comme la génération de code C++ qui est opérée après l'écriture du modèle (Fritzson et Engelson 1998). Enfin, les travaux menés par Adrian Pop à l'aide d'une démarche IDM (Pop 2008) ont conduit à la définition d'une extension de Modelica : MetaModelica.

Si Modelica offre des possibilités très intéressantes pour la modélisation multi-domaine, ce langage s'avère difficilement applicable dans notre contexte d'étude. En effet, la vision des physiciens vis-à-vis de leur objet d'étude est caractérisée par la mise en équation de chacun des composants du système. Ceci trouve son origine dans la bonne connaissance des lois qui régissent les composantes des systèmes complexes qu'ils soient mécaniques et/ou électriques. Les systèmes complexes en biologie de manière générale et en agronomie en particulier, sont plus difficilement décomposables en sous-unités dont le fonctionnement individuel pourrait être représenté sous forme d'équation. Les interactions entre les organes de la plante sont multifactorielles et peuvent intervenir à différentes échelles. Dans nombre de cas, des interactions ont été identifiées empiriquement sans qu'elles soient pour autant formalisables par un jeu d'équations. De plus, le choix retenu pour Modelica est qu'il n'y ait pas d'instanciation possible de nouveaux éléments au cours de la simulation. Ceci empêcherait la représentation de la structure de données décrivant le système plante-sol. Les tableaux définis

dans Modelica ne peuvent pallier ce défaut, puisque leurs dimensions doivent également être fixées au lancement de la simulation (Fritzson 2010b) or, dans le cadre d'un modèle grande culture, ces dimensions ne sont pas déterminées en amont (p.ex. le nombre de feuilles ou de ramifications créées par la plante). Enfin, Modelica reste un langage orienté objet exploitant les concepts d'héritage et de visibilité qui sont incompatibles avec l'objectif fixé de fournir aux agronomes un langage simple d'utilisation.

2.3.2- Ptolemy Project

Le projet Ptolémée trouve son origine à l'université de Berkeley. Initié au début des années 90, il a conduit à la sortie de Ptolemy Classic, Ptolemy Classic est environnement de modélisation hétérogène et de simulation développé en C++ se basant sur des formalismes visuels. La modélisation hétérogène consiste à permettre la coexistence au sein d'un même modèle de formalismes divers (p.ex. dataflow, machine à état, modélisation en temps continu). Sa dernière version (v.0.7.1) sortie en 1998 continue à être utilisée par différents acteurs mais n'est pas destinée à évoluer au sein du projet Ptolémée. Ce dernier a orienté son action vers le développement de Ptolemy II implémenté en Java (Eker *et al.* 2003). La composition de modèles hétérogènes hiérarchiques reste l'objectif majeur de cette nouvelle version de Ptolemy. Avec notamment comme finalité d'éviter les comportements émergents dus à une mauvaise gestion du branchement entre modèles hétérogènes. Dans le cadre de Ptolemy, la hiérarchisation des modèles et l'assemblage de modèles hétérogènes reposent sur le concept de modèle de calcul (MoC : pour *model of computation*) ainsi que la notion d'acteurs. Un MoC rassemble un ensemble de composants du modèle en assurant le contrôle du flot d'information ainsi que la modalité d'exécution de ses différents composants. Le MoC est lui-même considéré comme un composant, il peut donc être intégré dans un autre MoC. Ceci permet la hiérarchisation du modèle global.

Au niveau de l'interface de conception, les éléments de base sont des acteurs. Les acteurs communiquent entre eux via des ports, les ports pouvant être des entrées, des sorties ou jouer ces deux rôles en même temps. Les acteurs sont :

- soit atomiques, ils disposent alors d'une logique mathématique interne ;
- soit composites, leur rôle est alors de permettre l'agrégation d'acteurs.

Le MoC associé à l'acteur composite rassemblant l'ensemble des acteurs gouvernés par un même formalisme forment ce que les auteurs appellent un domaine. Les domaines de Ptolemy

correspondent aux grandes familles de formalismes de modélisation. Actuellement, les domaines existant dans Ptolemy et fournissant le MoC associé sont : les réseaux de processus (*process networks*), les modèles à événement discret, les dataflows dynamiques et dataflows synchrones, les modèles en temps continu, les machines à états finis. Certains domaines restent à l'état expérimental tel que le domaine des modèles à temps discret. La réception des informations par le port d'un acteur est gérée via un récepteur (*receiver)*. Le récepteur est déterminé par le domaine dans lequel l'acteur se trouve et injecté au moment de l'instanciation. Ceci permet de définir des acteurs polymorphes, c'est-à-dire qu'ils peuvent être utilisés sans modification dans des domaines différents.

Ptolemy couvre un large panel de formalismes de modélisation, son interface de conception visuelle de modèles est facile à manipuler et permet la simulation dans l'environnement de conception. L'objectif est de permettre la définition d'un prototype du modèle pour pouvoir ensuite procéder à la génération de son code dans un langage cible (C, VHDL,…) (Brooks *et al.* 2008b). La génération de code Java à l'aide de Copernicus a cessé d'être intégrée dans les versions récentes de Ptolemy II. Le processus de génération de code se base sur la conservation de la sémantique en fonction des éléments définis à l'aide de la syntaxe abstraite de Ptolemy telle qu'elle est définie dans (Lee 2010). S'il n'est que peu fait mention de démarche dirigée par les modèles (au sens IDM) par les chercheurs et développeurs, les concepts en sont pourtant bien présents. Autant sur l'analyse sémantique qui peut être menée sur les modèles, la gestion des syntaxes abstraites et concrètes ainsi que la possibilité pour le modélisateur de définir des transformations de modèles par réécriture de graphes (Brooks *et al.* 2008a).

Ptolemy offre donc un ensemble de fonctionnalités intéressantes quant aux objectifs fixés au chapitre 1. Il rend possible la définition de prototypes de modèles, d'en tester la validité, d'effectuer des vérifications formelles et d'obtenir un code spécifique à une plateforme technologique donnée. Malheureusement, peu d'intérêt est accordé aux modèles à temps discret qui restent à l'état expérimental depuis 2000 (Fong 2000). De plus, nous n'avons pas pu identifier de formalisme dans Ptolemy permettant de gérer une structure de données dynamique. Enfin, le nombre de formalismes offerts est très important par comparaison avec les besoins réels des modélisateurs agronomes. Ceci pourrait limiter l'acceptation de Ptolemy en tant qu'outil de modélisation.

2.4- Autres outils

2.4.1- DIESE

Nous nous sommes également intéressés aux travaux menés par Roger Martin-Clouaire et Jean-Pierre Rellier (Martin-Clouaire et Rellier 2009) autour d'une ontologie destinée à la modélisation et la simulation des systèmes de production agricole à l'aide d'un cadriciel : DIESE. Cette démarche a suscité notre intérêt puisqu'ontologie et metamodèle sont fortement liés (Bézivin et Gerbé 2001). En effet, Jean Bézivin définit un métamodèle comme « *A concrete representation of a shared conceptualization* » (Bézivin 2005a) ce qui rejoint la définition d'une ontologie telle que donnée par Thomas Gruber « *ontologies is an explicit specification of a conceptualization* » (Gruber 1995). Cependant, nous rejoignons le point de vue de Uwe Assman et de ses coauteurs (Assmann *et al.* 2006), pour lesquels une ontologie a une vocation descriptive au sens de Seidewitz (Seidewitz 2003) alors que le métamodèle a une vocation prescriptive. Nous reprenons ces points au chapitre 3 de ce mémoire et discutons la complémentarité de ces approches au chapitre 4.

Dans le cadre de DIESE, les aspects ontologiques semblent essentiellement dirigés sur la définition des pratiques agricoles et non sur les processus biophysiques. La conceptualisation d'un modèle est réalisée graphiquement avec l'interface Verdi suivant une syntaxe très proche de celle des diagrammes de classes UML (Ripoche *et al.* 2011). Ce choix semble peu adapté aux modélisateurs et dérivé des concepts connus des informaticiens. Il n'a pas été possible de tester les capacités de l'outil celui-ci n'étant pas en libre accès. Son orientation système de culture plus que prédiction de la croissance d'une culture ne paraît pas convenir à nos objectifs.

2.4.2- Workflows scientifiques

Certaines analogies transparaîtront entre la solution retenue dans le cadre de notre travail de thèse et les workflows scientifiques. L'utilisation de diagramme de blocs pour la conception visuelle n'y étant pas étrangère ainsi que la production de sorties à partir de fichiers de données. Plusieurs outils de gestion de workflows scientifiques existent, une revue des plus courants est dressée dans (Curcin et Ghanem 2008). Ces outils sont avant tout destinés à de l'analyse massive de données avec notamment des applications dans le domaine de la protéomique ou du séquençage (pour ne citer que des exemples biologiques) dans le cas de Pegasus (Deelman *et al.* 2005) qui permet la répartition des calculs sur des systèmes

distribués. Certains permettent également la mise en œuvre de modèles comme Kepler (Altintas *et al.* 2004), celui-ci est une extension de Ptolemy et s'appuie sur les mêmes concepts d'acteurs et de modèles de calcul. Cependant son utilisation à des fins de modélisation et simulation ne serait qu'un retour vers les fonctionnalités offertes par Ptolemy. Si l'analogie visuelle existe entre outils de gestion de workflow scientifique et la réalisation que nous exposons aux chapitres 3 et 4 de ce mémoire, il n'en va pas de même pour la sémantique sous-jacente et le domaine d'application auquel nous nous adressons.

3- Conclusion

Les outils de modélisation et de simulation généralistes ou spécifiques au domaine de l'agronomie ne permettent pas de répondre aux attentes définies au chapitre 1. L'obtention d'un environnement de modélisation conceptuelle de modèles agronomiques par une formalisation visuelle nous a paru possible par l'emploi des techniques de l'ingénierie dirigée par les modèles. Les DSMLs correspondent aux caractéristiques définies par (Robinson 2008) concernant la modélisation conceptuelle. De plus, en associant à des éditeurs graphiques permettant la modélisation conceptuelle des éditeurs basés sur une syntaxe textuelle, le DSML hybride ainsi obtenu pourrait conduire à la définition d'un environnement couvrant le cycle de modélisation conceptuelle et de prototypage d'un modèle agronomique. Pour que le prototypage de ce modèle soit mené à bien, le DSML doit fournir des fonctionnalités d'exécution au sein de l'environnement de conception. Cette exécution conduit à l'exploitation des résultats de la simulation par l'obtention de graphiques. Compte-tenu des pratiques des modélisateurs, des fonctionnalités de débogage sont également nécessaires. Enfin, l'obtention d'un code Java utilisable sur les plateformes de production d'ITK est envisageable à l'aide des transformations.

L'ensemble de ces démarches doit être réalisé avec application, les écueils à éviter sont nombreux. Selon (Kelly et Pohjonen 2009) le concepteur d'un DSML doit garder à l'esprit le délicat équilibre à trouver entre une trop grande généricité et une trop grande spécificité. La première mène à un langage trop imprécis et peu pratique d'utilisation par les experts du domaine. La seconde conduit à un langage dont le domaine d'application devient trop restreint ou dont le nombre de concepts est trop important. La multitude de formalismes offerts est alors difficile à appréhender par les experts. Le chapitre suivant expose la démarche que nous avons retenue pour mettre en place cet environnement de conception de modèles grande culture. Après une analyse de ce domaine, nous formalisons ses concepts

caractéristiques à l'aide d'un métamodèle, nous présentons ensuite le cadriciel servant de support à la génération de code et enfin nous donnons les éléments de spécification de la syntaxe concrète graphique du DSML ainsi que les éditeurs qui lui sont associés.

Chapitre 3 – Identification des concepts et proposition

Pour répondre à la problématique énoncée au chapitre 1, nous avons décidé de mettre en œuvre les technologies de l'IDM (Ingénierie Dirigée par les Modèles). L'objectif est l'obtention d'un environnement de modélisation et simulation (M&S) destiné aux agronomes. Cet environnement doit proposer une édition graphique des modèles pour les aspects de :

- hiérarchisation des modèles agronomiques ;
- flux d'information ;
- de séquence d'exécution.

L'orientation graphique de ces éditeurs est souhaitée pour se rapprocher des formalismes employés par les agronomes lors de la modélisation conceptuelle. De plus, elle devrait permettre de faciliter la compréhension d'un modèle lors de sa reprise par un agronome.

En ce qui concerne l'édition des expressions mathématiques et logiques représentant un processus biophysique, nous privilégions l'emploi d'un langage textuel préservant ainsi les pratiques des spécialistes du domaine.

La réunion des langages graphiques et textuels conduit à la notion de DSML hybride. Dans le cadre du travail de thèse, nous avons focalisé nos efforts sur l'obtention de la partie graphique de ce DSML. Comme nous l'avons vu au chapitre 2, deux composantes sont nécessaires à son obtention :

- sa syntaxe abstraite représentée par un métamodèle ;
- sa syntaxe concrète : ensemble d'éléments graphiques destinés à être manipulés par les modélisateurs.

La sémantique de ce langage n'est pas formalisée, comme exposé au chapitre 2, elle est associée à la notion d'exécutabilité du langage. Cette fonctionnalité ne pourra être mise en place qu'une fois le DSML défini pour les aspects graphiques et textuels. Cependant, pour faciliter la génération de code Java à partir de la conception graphique nous avons défini un cadriciel qui comporte pour partie la sémantique de notre langage.

Le DSML, ses éditeurs associés et la fonctionnalité de génération automatique de code font appel à la notion de fabrique logiciel (Greenfield et Short 2003). Nous avons donc choisi, pour l'environnement que nous cherchons à obtenir, la dénomination Crop Model Factory (CMF).

Au cours de ce chapitre, nous exposons les résultats de la démarche adoptée pour l'obtention des composantes majeures de CMF. Cette démarche correspond à celle présentée au chapitre 2 (figure 2.1.5). Nous avons donc procédé à l'analyse du domaine par l'étude d'un patrimoine applicatif : les modèles grandes cultures utilisés par ITK. A cette étude, s'est ajoutée une revue bibliographique des modèles grande culture (première partie). Grâce à cette analyse, nous avons défini un métamodèle (seconde partie), sa conception itérative a été menée en parallèle de l'implémentation du cadriciel support de la génération de code (troisième partie). Enfin, nous avons défini la syntaxe concrète graphique ainsi que spécifié les éditeurs qui lui sont associées (quatrième partie).

1- Analyse du domaine

La découverte du domaine de la modélisation grande culture a été menée de plusieurs façons. Elle nous a d'abord amené à une revue bibliographique la plus complète possible en l'état de nos connaissances autour du domaine de la modélisation en agronomie telle que présentée au Chapitre 1 de ce mémoire. Par la suite, nous avons souhaité réaliser la conception et l'implémentation d'un modèle de croissance (du blé) afin de nous imprégner des démarches du modélisateur et de mieux percevoir les contraintes d'implémentation d'un modèle de type grande culture. L'implémentation a été effectuée à la fois en Matlab et en Java, comme il était d'usage au sein d'ITK, et ceci a été réalisé en cherchant à comparer les différences inhérentes aux langages vis-à-vis de la factorisation des codes. Enfin, deux autres modèles d'ITK ont été étudiés pour mener des opérations de rétro-ingénierie et de réingénierie. Il s'agit d'un modèle de vigne, évoqué au Chapitre 1, ainsi que d'un modèle de croissance de coton. Pour ce dernier, la rétro-ingénierie n'a pu être menée à bien : en effet, son implémentation actuelle se base sur du C++, or sa redécouverte et sa ré-implémentation en Java (cible pour nos

développements professionnels) ont été chiffrées à six mois/homme, ce qui était incompatible avec les délais de cette thèse dans un contexte de financement CIFRE[15]. Cette partie présente donc le résultat du travail mené sur le modèle de blé ainsi que sur le modèle de vigne, elle se conclue sur les caractéristiques identifiées au sein des modèles « grande culture ».

1.1- Modèle de blé

En premier lieu, il nous faut préciser que ce modèle a été créé par anticipation sur un besoin ultérieur et donc sans que l'ensemble des objectifs de ce modèle soient complètement précisés[16]. Nous avons cependant retenu quelques éléments importants à prendre compte dans le modèle :

- estimation de la date de maturité des grains ;
- prédiction du rendement (masse de grains par unité de surface) ;
- prise en compte des effets du stress hydrique ;
- prise en compte de la disponibilité en matière azotée ;
- représentation plus ou moins détaillée des feuilles pour intégration ultérieure d'un modèle épidémiologique de maladies foliaires du type de la septoriose[17].

Une fois ces éléments connus, une revue bibliographique a été menée sur la physiologie du blé et les différentes représentations des phénomènes biophysiques paraissant important à intégrer (croissance foliaire, répartition de la matière carbonée et de la matière azotée, vernalisation). La majeure partie des formalismes retenus est issue de (Gate 1995; Jamieson *et al.* 1998; Jamieson et Semenov 2000; Lawless *et al.* 2005).

Classiquement, ce modèle est conçu sur la base du plant moyen, c'est-à-dire que tous les plants d'une parcelle donnée sont considérés comme se développant de la même manière. La figure 3.1.1 est une représentation schématique de la structure du plant de blé dans le cadre de ce modèle. Alors que le blé possède une tige principale et des ramifications secondaires appelées talles, le modèle ne considère qu'un seul axe. Conceptuellement cet axe est une fusion de la tige principale et des talles. Le modèle réalisé ne considère que la quantité de

[15] CIFRE : Conventions Industrielles de Formation par la Recherche

[16] Cet état de fait, en contradiction avec la notion d'intention du modélisateur, a compliqué la démarche de modélisation.

[17] La septoriose est une maladie d'origine fongique, le champignon se développe sur les feuilles entrainant leur nécrose. Son action a un impact sur le rendement des cultures.

matière azotée et de matière carbonée dans la tige alors que la hauteur et la largeur de celle-ci ne sont pas représentées. A l'instar du modèle Sirius (Jamieson *et al.* 1998), la représentation des feuilles est stratifiée. Chaque strate regroupe l'ensemble des feuilles de même âge, la strate est considérée comme une seule grande feuille. Elle est caractérisée par une masse de matière organique carbonée, une concentration en matière azotée, par sa surface et des paramètres régissant l'évolution de sa surface au cours du temps. Enfin, les grains sont regroupés dans un unique compartiment caractérisé par leur masse totale et leur composition (azote/carbone).

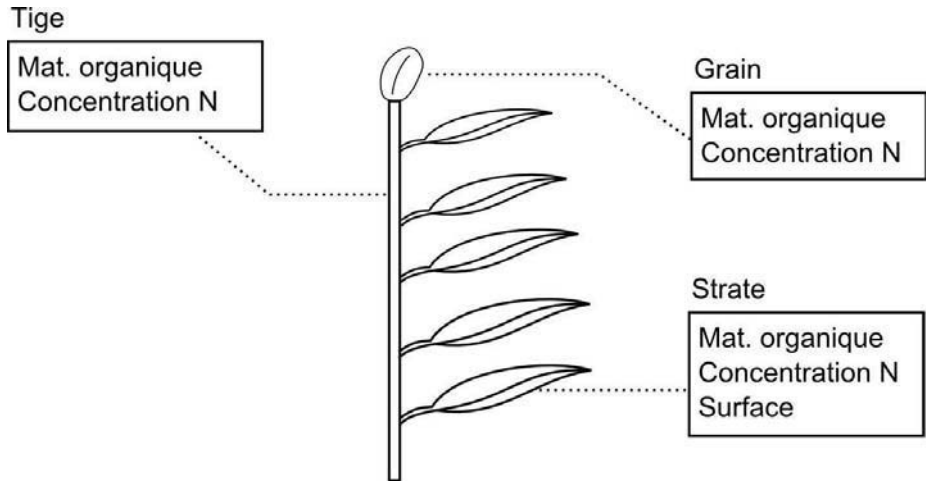


Figure 3.1.1 : Représentation schématique du plant de blé tel qu'intégré dans le modèle.

La dynamique de cette structure est régie par les processus biophysiques représentés dans le modèle conceptuel de la figure 3.1.2. Il n'existe pas de modalité formelle d'expression du modèle conceptuel (cf. Chapitre 1), il s'agit d'une expression libre dont la vocation est d'aider le modélisateur dans sa démarche de conception. Les formalismes que nous avons adoptés se focalisent sur la séquence d'exécution du modèle, c'est-à-dire l'ordre dans lequel interviennent les différents sous-éléments du modèle durant la simulation. Cet ordre est déterminé par les flèches qui relient les processus (les rectangles) ou les expressions conditionnelles (les triangles). Le point de départ, le premier sous-élément à intervenir à la première itération de la simulation, est classiquement situé en haut à gauche du modèle conceptuel. Il s'agit d'une évaluation conditionnelle qui dépend de la mise en place de

l'initiation florale[18]. Celle-ci est conditionnée par la vernalisation qui correspond à l'exposition du plant de blé à de basses températures après la germination. Pendant cette phase de la simulation, le nombre de feuilles mises en place est recalculé jusqu'à ce que l'initiation florale soit atteinte, le nombre de feuilles final est alors fixé.

Que l'initiation florale soit atteinte ou non, le développement[19] de la plante a lieu, l'avancement sur l'échelle phénologique est effectué grâce à un cumul de degrés/jour. Le passage d'un stade phénologique au suivant est déterminé par une valeur seuil spécifique à la variété de blé considérée. Une fois le stade « émergence » atteint, débute la mise en place des feuilles. Celles-ci conduisent à la prise en compte de la photosynthèse, permettant alors de considérer l'augmentation de surface foliaire et, le cas échéant, le remplissage des grains.

Bien que présents à l'esprit du modélisateur, les transferts d'informations entre processus ne sont pas représentés dans le modèle conceptuel, de même que l'échelle temporelle ou l'interaction avec la structure représentant la plante. Comme écrit au chapitre 1, les expressions mathématiques décrivant le système dynamique ne sont pas formalisées, il peut y être fait référence par l'association d'un processus donné à une publication scientifique qui en propose une formalisation mathématique. Plus souvent, les équations sont définies dans la documentation en langage naturel associée au modèle conceptuel, c'est également dans ce document que seront décrits les paramètres et variables d'état intervenant dans cette équation ainsi que les données externes (les entrées) nécessaires à son exécution. Nous ne présenterons pas plus avant les détails de conception et d'implémentation du modèle blé. Ce modèle a permis de nous familiariser avec les concepts présents dans les modèles « grande culture ». L'étape suivante s'est focalisée sur le modèle vigne dont une partie a déjà été exposée dans le chapitre 1. Nous allons présenter, brièvement les éléments de réflexion qu'a fournis la rétro-ingénierie du modèle vigne.

[18] Après l'initiation florale, les méristèmes cessent de produire des feuilles pour mettre en place des inflorescences. L'organogénèse se détermine tôt dans le cycle de développement de la plante, l'initiation florale se déroule avant même que les premières feuilles ne soient visibles à l'œil nu.

[19] Vu au chapitre 1, le développement de la plante est lié à la phénologie.

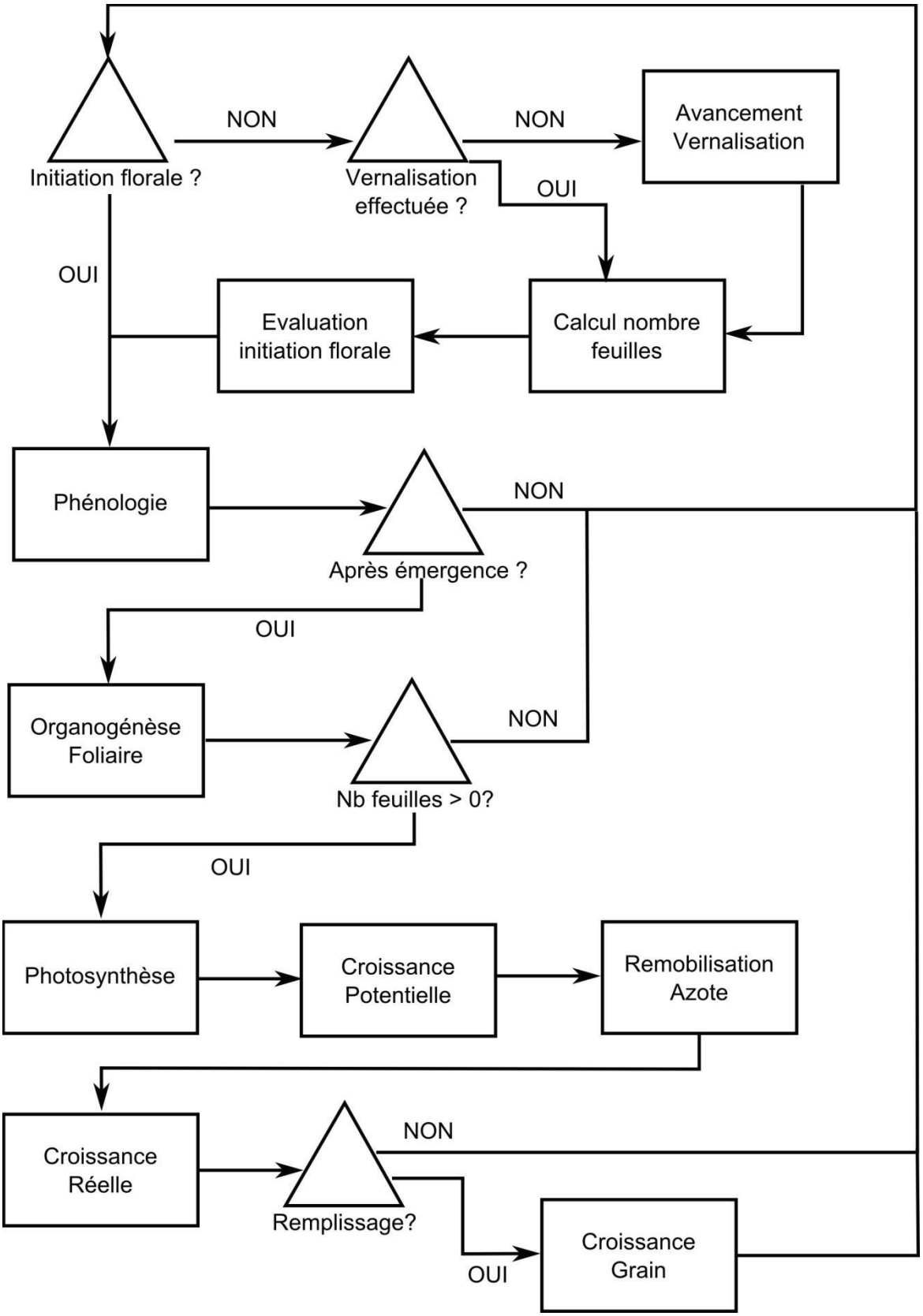


Figure 3.1.2 : Modèle conceptuel décrivant le modèle blé suivant un formalisme libre

1.2- Modèle de vigne

Dans un premier temps, la découverte du modèle vigne a été menée par l'étude de son implémentation en Java, de deux diagrammes de classes concernant le simulateur ainsi que du document en langage naturel décrivant le modèle. Plusieurs constatations se sont rapidement imposées. Les diagrammes de classes, bien que fournis (environ 70 classes), ne permettaient pas de comprendre la logique de l'implémentation, de plus, leur mise à jour n'avait pas été réalisée. L'implémentation elle-même s'est avérée difficile à appréhender : dans le contexte historique du projet qui a abouti à cette implémentation, celle-ci a été menée sans maitrise réelle des concepts orientés objets. Nous avons ensuite mené une comparaison entre l'implémentation Java et le prototype en Matlab. Celle-ci a montré des disjonctions entre les deux modèles, confortant ainsi l'analyse menée sur le risque posé par le processus de transcription (cf. Chapitre 1).

La découverte du modèle n'a pas donné lieu à une rétro-conception, elle fait apparaître la nécessité de procéder à une réingénierie du modèle qui a conduit à la redéfinition du modèle conceptuel. Celui-ci a été présenté, pour partie, dans le Chapitre 1. Certains éléments en sont absents. En effet, le simulateur associé au modèle vigne présente plusieurs configurations qui ont pour objectif de tester différentes représentations des processus et du système plante-sol. La construction d'un modèle grande culture est un processus itératif, dans certains cas, il est difficile de choisir *a priori* le formalisme le plus adapté pour un processus biophysique donné. L'intégration de différents formalismes dans le prototype et la comparaison de leurs performances respectives vont fournir au modélisateur des critères objectifs quant à la définition du modèle le plus adapté au contexte d'étude. Pour la plante, deux configurations existent avec des implications sur les processus représentant sa croissance :

- une représentation parallélépipédique du rang de vigne (cf. Chapitre 1) ;
- une représentation explicite de la topologie de la plante. La topologie est définie par une succession de phytomères de différents types, leur mise en place est contrôlée par un processus semi-markovien (Louarn 2009).

Pour le sol, deux représentations existaient et une troisième était en cours de préparation lorsque le travail a été mené :

- le sol comme un unique réservoir (cf. Chapitre 1) ;
- une décomposition du sol en différents horizons, les transferts d'eau par infiltration entre ces différents horizons étant représentés ;
- une décomposition du sol en vertex (trois dimensions), prenant en compte les transferts d'eau horizontaux et verticaux.

Ces différentes configurations ont pour objet de déterminer le modèle le plus pertinent compte-tenu des objectifs fixés. Elles sont avant tout des outils de recherche mais ne sont pas destinées à être utilisées dans l'outil d'aide à la décision final, seule la configuration la mieux adaptée en a la vocation. Cet élément est important à prendre en compte dans le cadre de la mise en œuvre d'un outil de conceptualisation de modèles, objet de cette étude.

La réingénierie de l'implémentation Java a été menée dans le cadre de la définition du cadriciel de simulation (voir ci-après 2ème partie). La rétro-ingénierie menée sur le modèle vigne est restée informelle. Cependant, associée à la construction du modèle blé et à l'étude bibliographique des modèles « grande culture », elle a permis d'identifier les éléments caractérisant ce type de modèle. Dans la partie suivante, nous présentons l'axe retenu pour la caractérisation des concepts clés de ce domaine ainsi que le résultat de l'analyse de ce domaine.

1.3- Caractérisation des concepts

1.3.1- Axe de caractérisation retenu

L'identification des concepts aurait pu être menée suivant deux axes différents que la figure 3.1.3 vise à expliciter. La première possibilité (partie de droite) correspond d'abord à l'identification des phénomènes biophysiques, communément mis en œuvre dans un modèle grande culture, tels que l'interception lumineuse ou encore l'évapotranspiration potentielle. Après l'identification des processus, suit un inventaire des différents formalismes connus pour chacun d'eux. Cette première approche est descriptive, elle vise à dresser un inventaire des connaissances du domaine, nous la qualifions d'approche ontologique, rejoignant ainsi la définition des ontologies telle que proposée par (Assmann *et al.* 2006). Le deuxième axe consiste à considérer l'ensemble des représentations possibles pour les phénomènes biophysiques ainsi que leur assemblage, cet ensemble correspond à l'ensemble des modèles

grande culture. En reprenant l'approche ensembliste de (Favre 2004a), il s'agit donc du langage d'expression des modèles grande culture pour lequel il est possible de définir un métamodèle. Dans ce cas, il est possible d'orienter la démarche sur les caractéristiques communes des processus indépendamment de leur signification biologique. Pour résumer, l'approche ontologique consiste à caractériser les éléments représentés dans un modèle agronomique alors que l'approche par métamodélisation vise à caractériser les éléments de représentation permettant de formaliser le contenu d'un modèle grande culture.

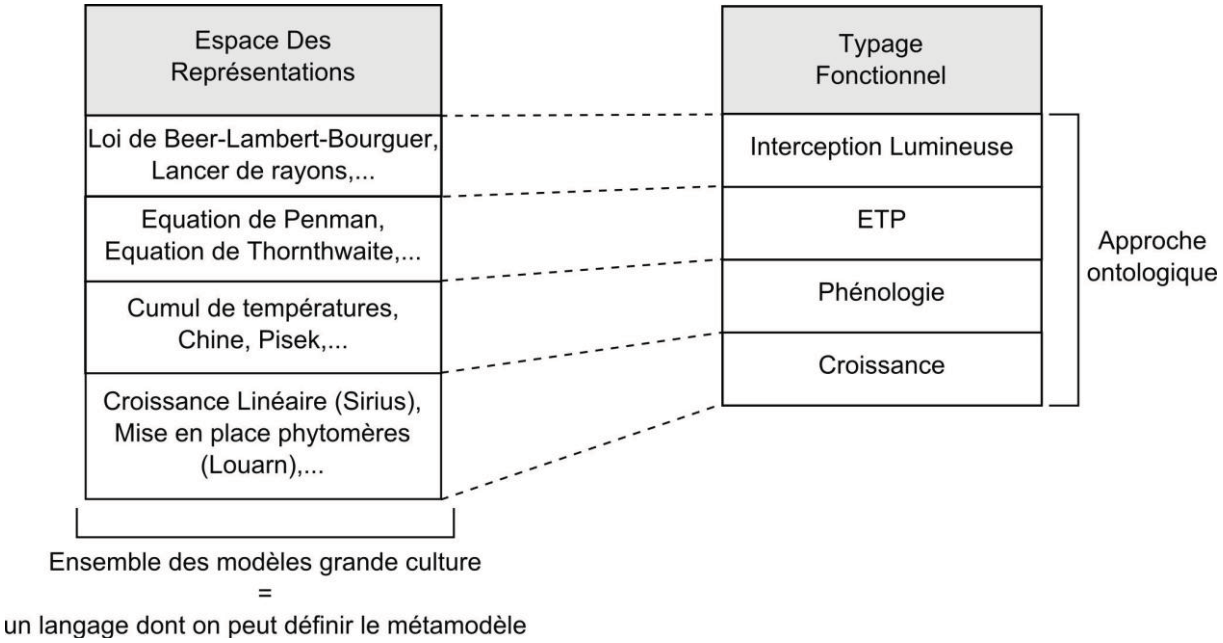


Figure 3.1.3 : Les deux axes possibles pour la caractérisation des concepts inhérents aux modèles grande culture

Concernant l'approche ontologique, elle revient à définir une bibliothèque de modélisation à partir de laquelle différentes représentations pourraient être extraites et assemblées afin de définir un modèle. L'assemblage est une notion forte dans ce contexte, elle fait appel à la composabilité des différents éléments. Telle que l'évoquent Rizzoli et ses coauteurs (Rizzoli *et al.* 2007), la mise en place de composants dans le cadre de la M&S pour des domaines comme celui de la micro-électronique est tout à fait réalisable mais elle s'avère beaucoup plus délicate pour les systèmes biologiques complexes. En effet, la diversité de formalismes existants pour un phénomène donné peut rendre l'approche par composants contraignante. Par exemple, la représentation de l'interception lumineuse suivant la loi de Beer nécessite comme

information le Leaf Area Index[20] et le rayonnement global, alors que le lancer de rayon utilise une représentation en trois dimensions du couvert végétal et l'angle d'incidence des rayons. Ces deux représentations permettent de calculer le rayonnement absorbé par le couvert végétal. Dès lors se pose la question de l'uniformisation par l'approche composant de ces deux représentations afin de les rendre interchangeables. Des recherches ont été menées dans ce sens dans le domaine de l'écologie (Rizzoli *et al.* 2008) ou de la modélisation en agronomie (Hillyer *et al.* 2003), cependant les solutions proposées reposent sur un réel savoir-faire en génie logiciel et ne peuvent être facilement mises en œuvre par les modélisateurs agronomes. Outre les difficultés techniques qui découlent de cette approche, la définition d'une ontologie de ce type peut également limiter les potentiels futurs de la solution qu'elle aura servi à définir. En effet, avec l'évolution des connaissances, il est possible qu'il soit nécessaire d'intégrer de nouveaux processus dans la démarche de modélisation. L'approche ontologique ne faciliterait pas cette intégration.

Que ce soit sur les problématiques de branchement de processus ou sur l'intégration de nouveaux processus, nous avons pour objectif de définir un environnement qui permette au modélisateur agronome de garder la plus grande latitude possible quant à la définition de son modèle. C'est par l'approche par métamodélisation, s'affranchissant des considérations biologiques, que nous pensons pouvoir y parvenir en définissant les éléments qui constituent un modèle grande culture, les règles qui régissent leur assemblage, ainsi que les invariants de ce type de modèle et sa modalité d'exécution. La réalisation d'une ontologie n'est cependant pas incompatible avec notre démarche et offre des perspectives intéressantes sur lesquelles nous reviendrons dans le dernier chapitre de ce mémoire.

1.3.2- Les concepts identifiés

La démarche adoptée s'est d'abord focalisée sur la caractérisation des processus biophysiques présents dans les modèles grande culture. La figure 3.1.4 représente les éléments d'importance qui les composent. Un processus a pour vocation de produire de l'information sous la forme d'une ou plusieurs sorties numériques. Pour cela, il exploite des informations numériques en tant qu'entrées. Ces informations sont issues des sorties d'autres processus ou sont des données environnementales fournies pour les besoins de la simulation. Les entrées sont transformées suivant un algorithme composé d'expressions logiques et d'expressions mathématiques (cf. Chapitre 1). Cet algorithme fait appel à des paramètres et à d'éventuelles

[20] Le LAI correspond à la surface foliaire totale par unité de surface au sol.

variables d'état. La lecture des entrées et la production de sorties s'effectue selon une rythmicité donnée, celle-ci est déterminée par le pas de temps défini pour le processus. Au sein d'un même modèle agronomique peuvent coexister des processus définis à des pas de temps différents, cependant le plus petit pas de temps est un dénominateur de chacun des pas de temps présents dans le modèle.

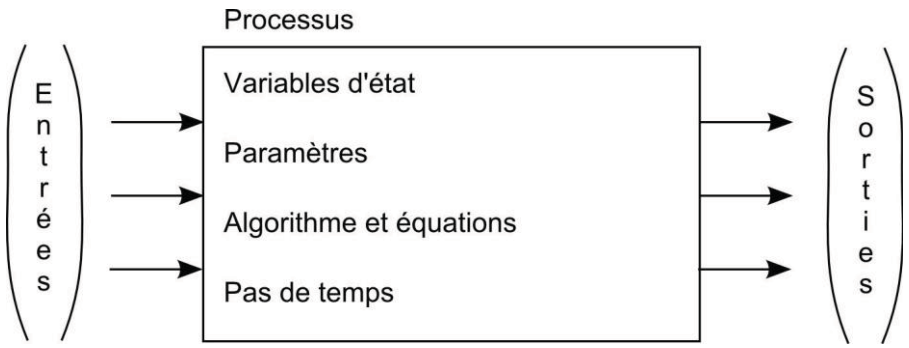


Figure 3.1.4 : Représentation des éléments de caractérisation des processus des modèles grande culture.

Si les processus possèdent systématiquement des paramètres, il n'en va pas de même pour les variables d'état. Celles-ci ont pour vocation de conserver en mémoire un état du système au pas de temps précédent. Il est possible qu'une variable d'état soit exclusivement lue et mise à jour par un unique processus, mais cela reste un cas de figure assez rare. En effet, les variables d'état sont bien souvent partagées entre différents processus, elles sont regroupées dans une structure d'information décrivant l'état du système plante-sol à un pas de temps donné. Dans leur article (Dalle *et al.* 2009), Dalle et ses coauteurs désignent un espace où tous les processus peuvent lire ou écrire des variables d'état comme étant un tableau noir (*blackboard*), cette terminologie est issue de l'intelligence artificielle (Engelmore et Morgan 1988). Dans le cadre de la modélisation en agronomie, une particularité d'importance existe. En effet, certains processus ont pour vocation de faire évoluer la structure même de l'information, ce sont les processus responsables de l'organogénèse. Ceux-ci vont avoir pour rôle de créer un nouvel organe, par exemple une feuille. Cet organe est caractérisé par un ensemble de variables d'état, comme sa surface, son âge ou encore sa concentration en éléments azotés. L'existence de cette structure et sa dynamique régie par les processus sont importantes à prendre en compte dans notre démarche.

En ce qui concerne les modalités d'exécution de la simulation, plusieurs points sont à noter. Comme nous l'évoquions au premier chapitre, la simulation s'exécute systématiquement de date à date, c'est-à-dire avec un pas de temps apparent de vingt-quatre heures. Les processus

sont exécutés de façon séquentielle suivant l'ordre établi lors de la modélisation conceptuelle du modèle agronomique. La coexistence de plusieurs processus ayant des pas de temps différents a des conséquences. En effet, suivant le pas de temps lors de la simulation, la séquence de processus effectivement exécutée va varier. De plus, l'émission de sorties par un processus à un rythme différent du rythme du processus qui les utilise en tant qu'entrées implique une logique de branchement de modèles.

Ces éléments, directement issus de l'analyse du domaine, sont des concepts indispensables pour la définition du DSML (Domain-Specific Modeling Language). Leur identification s'est suivie de la formalisation de la syntaxe abstraite du DSML via un métamodèle. La partie suivante de ce chapitre, présente le résultat de cette démarche itérative : le métamodèle C3M pour *Crop Models MetaModel*.

2- Le métamodèle C3M

C3M a été défini selon une démarche itérative, des versions successives ont fait l'objet de publications(Barbier *et al.* 2011; Barbier *et al.* 2012; Barbier *et al.* 2013a - In press, 2013b - In press). Le cadriciel a été défini en parallèle de C3M, l'abstraction nécessaire à la conception du cadriciel a permis de tester les concepts intégrés dans C3M et d'en évaluer les limites. Les limites identifiées ont conduit à des évolutions du métamodèle. Un sous-ensemble du cadriciel est intégré dans un chapitre de livre à paraître (Barbier *et al.* 2013a - In press). C3M a été conçu en gardant à l'esprit la nécessité de permettre à terme la réutilisation d'éléments de modélisation d'un modèle grande culture à un autre. Ce mémoire présente le contenu de C3M au temps de la rédaction, il est vraisemblable que l'ajout de fonctionnalités à CMF amène à des évolutions de son métamodèle. Certaines évolutions sont déjà envisagées et présentées au Chapitre 5. Dans le cadre du travail de thèse, la conception de C3M s'est focalisée sur la mise en place de la syntaxe abstraite support de la partie visuelle du DSML. Des travaux supplémentaires seront nécessaires pour la définition de la partie textuelle.

2.1- Modèles et séquence d'exécution

La figure 3.2.1 représente, sous la forme d'un diagramme de classes, la partie[21] du métamodèle dédiée à la représentation d'un processus. Nous avons retenu la dénomination de modèle atomique à la manière de DEVS. Le modèle atomique est la plus petite subdivision du modèle conceptuel en deçà de laquelle le modélisateur fait intervenir algorithme et équations mathématiques. Le modèle atomique est une traduction directe du processus tel qu'il est représenté dans la figure 3.1.4. Sa logique interne n'est, pour le moment, pas représentée. Il est composé d'entrées, de sorties, de variables d'état et de paramètres. Tous ces éléments ainsi que le modèle atomique possèdent un attribut *description* ayant pour vocation de permettre la documentation du modèle grande culture. La méthode *execute* du modèle atomique est destinée à être appelée à un rythme défini par l'attribut *timestep*.

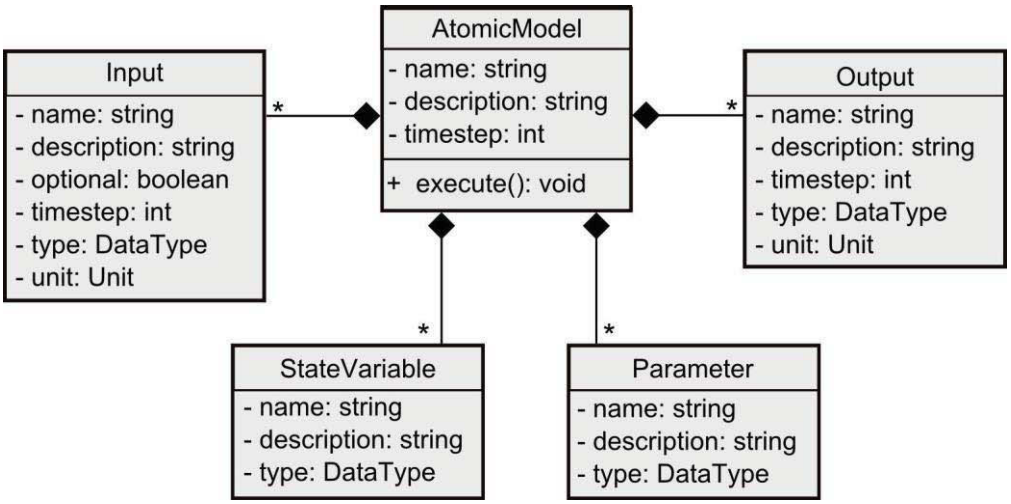


Figure 3.2.1 : Représentation, sous forme de diagramme de classes, de la partie du métamodèle concernant les modèles atomiques

A partir de cet élément conceptuel de base, le métamodèle a été enrichi afin d'intégrer la notion de hiérarchie, de séquence d'exécution et de séquence d'exécution alternative. Enfin le concept de modèle atomique a été adapté afin de prendre en compte des processus intervenant de manière différente dans la séquence d'exécution. Le résultat de l'évolution du métamodèle est produit dans la figure 3.2.2.

La hiérarchisation du modèle grande culture est mise en œuvre à l'aide du patron de conception composite (Gamma *et al.* 1995). Le modèle composite (*CompositeModel*) est

[21] Pour des raisons de lisibilité il paraît préférable de fournir plusieurs sous-ensembles du métamodèle, le diagramme complet est fourni en annexe de ce mémoire.

composé d'un ensemble de modèles et possède une référence vers le premier modèle de sa séquence d'exécution (relation *firstInFlow*). Tout modèle possède une référence pointant sur son successeur dans la séquence d'exécution ou alors cette dernière est vide si le modèle est le dernier de la séquence. Comme nous l'avons vu pour le modèle blé (Figure 3.1.2), des conditions peuvent être intégrées au modèle conceptuel qui, suivant l'état du système, vont amener à suivre une séquence différente. Ce cas de figure existe aussi pour le modèle vigne avec une particularité, dans certains cas, la condition n'est pas liée à un état ou un changement d'état du système mais à une configuration du modèle qui est déterminée au lancement de la simulation (cf. ci-avant, partie 1.2). Pour permettre ce type de construction, nous avons intégré au métamodèle un type de modèle particulier, l'*AlternateSequenceModel*, dont le rôle est de définir une alternative. Cette alternative est conditionnée à une expression logique, si cette expression est vraie la séquence se poursuit suivant la relation *nextInFlow*, si elle est fausse la séquence se poursuit suivant la relation *alternateInFlow*. Cette construction permet de facilement intégrer des séquences alternatives à la fois dans la hiérarchie de modèles et dans leur séquence en n'introduisant qu'une seule relation sémantique spécifique. Si, pour un modélisateur, ce concept d'alternance de séquence est nécessaire, il pourrait lui apparaître surprenant que ce concept soit porté par un élément de type *Model*. Afin d'éviter toute confusion, ce lien sémantique ne doit pas transparaitre lors de la définition de la syntaxe concrète du DSML (à savoir la représentation graphique).

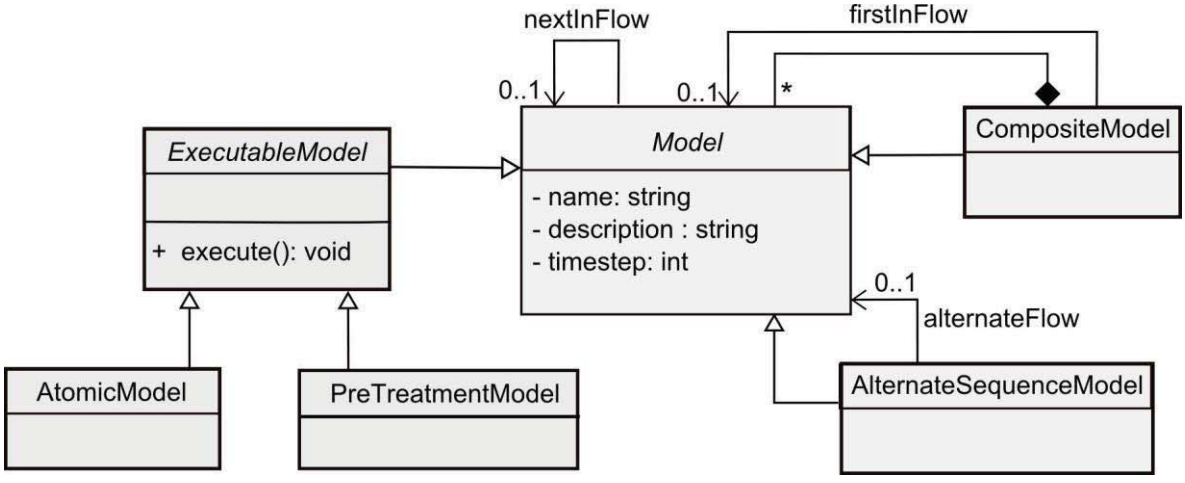


Figure 3.2.2 : Les différents éléments du métamodèle permettant de gérer la hiérarchisation du modèle agronomique ainsi que la séquence d'exécution et ses particularités

Enfin, la présence potentielle de divers pas de temps dans un même modèle grande culture nous a mené à la refonte du concept de modèle atomique. Celui-ci appartient désormais à la

catégorie des modèles exécutables (*ExecutableModel*), tout comme le modèle de prétraitement (*PreTreatmentModel*). La différence entre ces deux types de modèles exécutables tient dans leur modalité d'intervention dans la séquence d'exécution.

La figure 3.2.3 illustre par un cas concret la nécessaire intégration de ces deux concepts. La partie supérieure de cette figure représente une séquence de quatre processus telle qu'elle pourrait être formulée dans un modèle conceptuel. Le premier d'entre eux évalue le rayonnement absorbé par la plante à un pas de temps horaire. A partir du rayonnement absorbé, le second détermine la perte d'eau potentielle de la plante par transpiration suivant un pas de temps horaire, ceci lui permet également de déterminer la pression de succion racinaire[22]. Cette pression est nécessaire à la réalisation, par le troisième processus, du bilan hydrique du sol grâce à l'équation de Richards (Richards 1931). Ce bilan est effectué à un pas de temps de l'ordre de la seconde[23]. Il permet d'évaluer la quantité d'eau effectivement disponible dans le sol et, donc, la détermination de la transpiration réelle par le quatrième processus suivant un pas de temps horaire. Compte tenu des pas de temps respectifs de ces processus, la simulation de ce modèle se déroule avec un pas de temps d'une seconde. L'échelle de temps, au bas de la figure, représente une heure de temps de simulation. Pour chaque pas de temps, les processus exécutés sont représentés par un engrenage. Suivant la logique définie ci-avant, l'interception lumineuse et la transpiration potentielle sont exécutées au premier pas de temps de l'heure de simulation, alors que la transpiration réelle n'est exécutée qu'au dernier pas de temps de l'heure de simulation. Ces trois processus sont définis à un même pas de temps et ne sont pourtant pas exécutés au même temps de la simulation. C'est l'existence d'un pas de temps effectif inférieur à l'heure qui permet d'observer cette distinction. Les processus évaluant interception et transpiration potentielle, exécutés en début de plage horaire, sont considérés comme des modèles de prétraitement. Alors que la transpiration réelle, intervenant en fin de plage horaire, est un modèle atomique.

[22] La succion correspond à une pression négative permettant à la racine d'absorber de l'eau présente dans le sol à condition que la valeur absolue de la succion racinaire soit supérieure à celle du sol.

[23] Ce pas de temps dépend de l'épaisseur des horizons de sol retenue par le modélisateur.

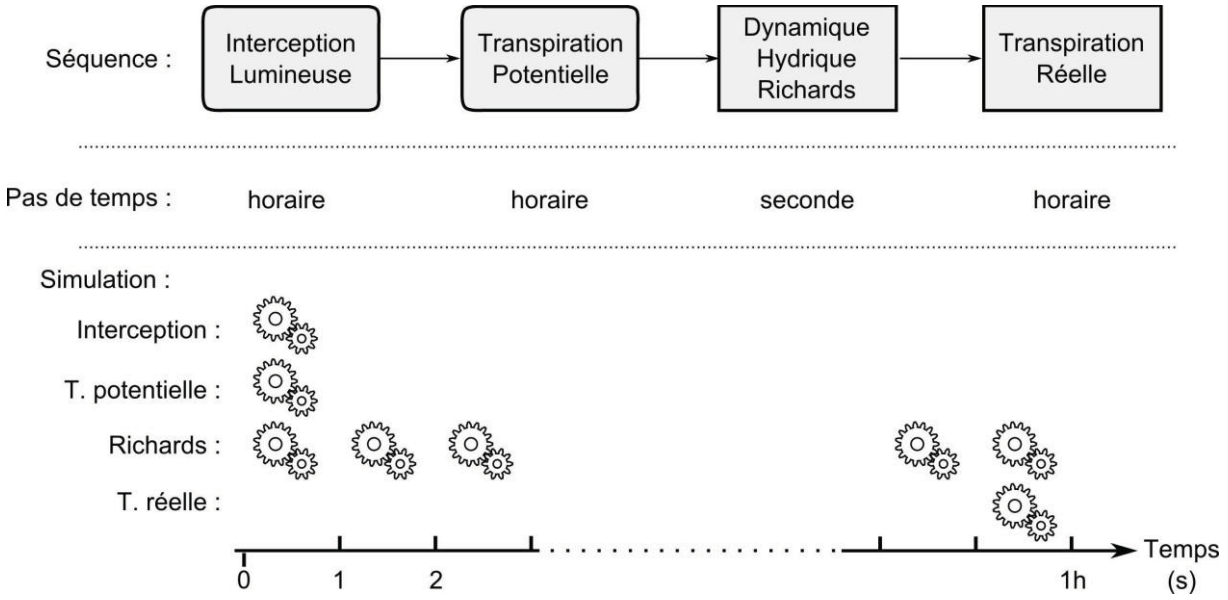


Figure 3.2.3 : Modèle conceptuel simple et représentation du déroulement de la simulation correspondante justifiant l'intégration des modèles de prétraitement au métamodèle C3M

Cette première partie de la présentation de C3M s'est focalisée sur la présentation des entités modèles permettant de gérer la hiérarchisation du modèle agronomique, la représentation des processus ainsi que la séquence d'exécution de la simulation. Nous allons maintenant aborder la représentation des données dans C3M, nous verrons que la possible existence de différents pas de temps dans le modèle agronomique a des conséquences sur la transmission des informations.

2.2- Entrées, sorties et adaptation

La figure 3.2.1 donnait déjà un aperçu des données telles qu'elles sont formalisées dans C3M en représentant entrées, sorties et variables d'état des modèles exécutables. La figure 3.2.4 en livre une version exhaustive.

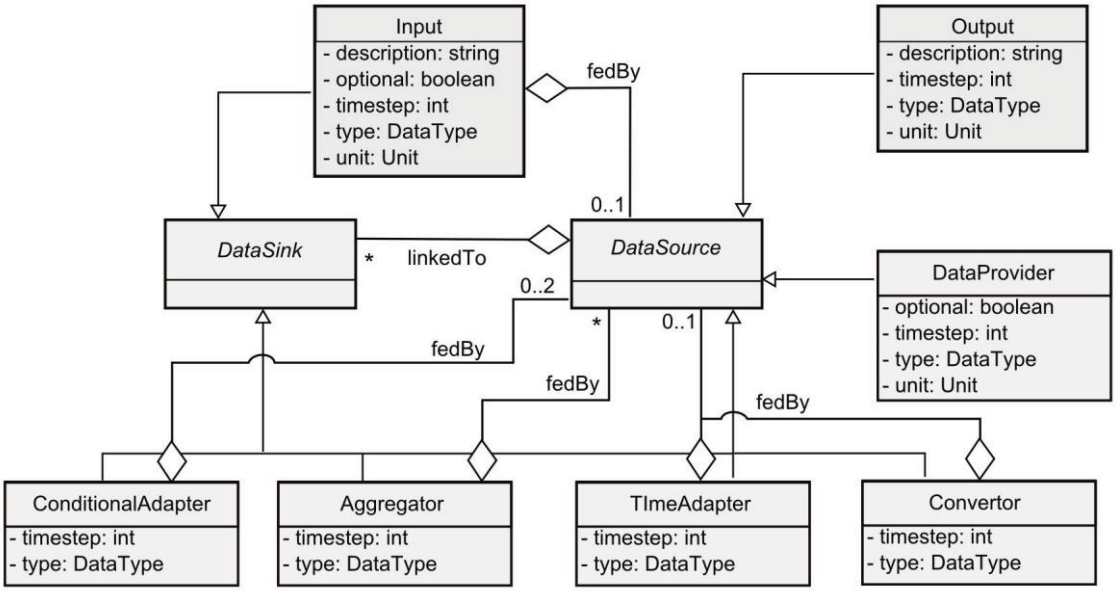


Figure 3.2.4 : Partie du métamodèle C3M présentant les échanges de données à l'aide des concepts de source et de puits de données

Cette construction s'est basée sur le fait que des éléments du modèle produisent de l'information et jouent donc le rôle de source de données (*DataSource*). Ce rôle est assumé par les sorties des modèles (*Output*) ainsi que par les fournisseurs de données (*DataProvider*). Ces derniers permettent d'alimenter la simulation en informations environnementales (p. ex. rayonnement solaire, précipitations) qui sont nécessaires à sa bonne exécution. Une source de données peut être reliée à un nombre indéterminé d'éléments consommant l'information produite : des puits de données (*DataSink*). L'entrée d'un modèle (*Input*) est l'exemple typique d'un puits de données.

Un ensemble de concepts présents dans C3M jouent le double rôle de puits et de source de données, il s'agit des adaptateurs. Les adaptateurs sont la solution que nous avons employée pour répondre à la problématique du branchement de modèles dans un contexte de réutilisation des modèles et de coexistence de pas de temps différents dans un même modèle agronomique. Quatre types d'adaptateurs semblent nécessaires dans le cadre de la conception des modèles grande culture : un adaptateur temporel (*TemporalAdapter*), un agrégateur (*Aggregator*), un convertisseur (*Convertor*) et un adaptateur conditionnel (*ConditionalAdapter*).

Les fonctions de ces adaptateurs sont les suivantes :

- L'adaptateur temporel a pour objectif de gérer la problématique de branchement entre une source et un (ou plusieurs) puits fonctionnant à des pas de temps de différents. Si plusieurs puits sont concernés par la liaison, ils doivent consommer l'information à la même fréquence. La règle par défaut est l'addition (respectivement la division par équirépartition) lorsque le pas de temps de la source est inférieur (respectivement supérieur) à celui du puits ;
- L'agrégateur permet de fusionner plusieurs sources d'informations en une seule. Prenons l'exemple de l'évapotranspiration réelle, en fonction de la décision du modélisateur, elle peut être calculée et transmise comme une seule information ou alors être calculée et produite suivant deux sorties distinctes : la transpiration réelle et l'évaporation réelle. Dans ce dernier cas de figure, le branchement avec une entrée représentant l'évapotranspiration réelle nécessite l'utilisation intermédiaire de l'agrégateur qui permettra d'additionner évaporation réelle et transpiration réelle. La probabilité de rencontrer ce cas de figure est augmentée dans un contexte de réutilisation des processus d'un modèle agronomique à un autre ;
- Le convertisseur répond lui aussi à une problématique dont la probabilité d'apparition augmente avec la réutilisation. En effet, il permet de gérer la représentation d'une même grandeur physique par différentes unités. Le système international est encore difficilement adopté dans le domaine de l'agronomie, il est donc indispensable de permettre la mise en œuvre d'un convertisseur ;
- L'adaptateur conditionnel, quant à lui, permet d'alimenter un ou plusieurs puits de données en changeant l'origine de l'information (la source effective). Par exemple, certaines stations météorologiques fournissent l'évapotranspiration potentielle (ETP) d'autres ne sont pas dotées des éléments de mesure nécessaires. Si un modèle atomique nécessite la fourniture de l'ETP en entrée, suivant les conditions, cette information pourra être obtenue à partir d'un fournisseur de données externes ou de la sortie d'un modèle dédié au calcul de l'ETP. L'adaptateur conditionnel permettra d'alterner entre ces deux sources de données.

Le partie de C3M que nous venons de présenter permet de gérer le flux d'informations direct exploité par les différents modèles exécutables présents dans un modèle grande culture. Comme évoqué ci-avant (cf. partie 1.3.2 de ce chapitre), il ne s'agit que d'une partie des données intervenant dans les modèles grande culture. En effet, la structure représentant le

système plante-sol est une construction essentielle de ces modèles. Les modèles exécutables sont amenés à lire et/ou à écrire des informations dans cette structure et, pour certains d'entre eux, à lui donner une dynamique. La partie suivante présente la portion du métamodèle dédiée à cet ensemble de variables d'état que nous avons désigné par le terme *blackboard*, par analogie avec l'appellation donnée dans (Dalle *et al.* 2009).

2.3- Le *Blackboard*

Le *blackboard* est une structure partagée par l'ensemble des modèles exécutables, il contient à la fois une représentation des différentes structures présentes dans le système plante-sol et des variables permettant de représenter l'état de ces structures. Lors des premières réflexions menées sur cette représentation (Barbier *et al.* 2011), avait été envisagé d'imposer la décomposition du système plante-sol (Figure 3.2.5). Cette représentation du système est une transposition directe de ses composantes suivant le paradigme objet. Il s'agit de la représentation la plus détaillée possible à partir de laquelle il serait possible de formaliser des représentations plus simples telles que pour les modèles big leaf. Cependant, elle nous paru poser les mêmes problèmes d'évolution que ceux posés par l'approche ontologique pour décrire la dynamique du système (cf. ce chapitre 1.3.1). De plus, son utilisation pour représenter des systèmes plus simples aurait obligé les modélisateurs à manipuler des concepts de manière peu naturelle. En effet, pour représenter une plante dans le cadre d'un modèle big leaf, le modélisateur doit, en principe, définir une structure feuille et un éventuel compartiment grains. Cette représentation est tout fait possible en utilisant le diagramme de classes de la figure 3.2.5. Cependant, il faudrait instancier : un champ, une population, un plant, une tige et un phytomère, avant de pouvoir instancier la classe qui intéresse le modélisateur : la feuille. Cette situation amènerait donc à spécifier un grand nombre d'éléments inutiles par rapport à l'objectif de représentation, elle risquerait donc d'être source d'insatisfaction pour les modélisateurs.

Nous nous sommes donc orientés sur une formalisation plus polyvalente du *blackboard* tout en étant plus simple en termes de notations (Figure 3.2.6). La décomposition structurelle est réalisée à l'aide de *StructuralElement*, chacune de ces entités contenant potentiellement des variables d'état. Cette formulation très générique peut paraître un peu obscure, son utilité sera précisée dans la partie 3 de ce chapitre abordant la syntaxe concrète du DSML. Comme exposé par (Kelly et Pohjonen 2009), le concepteur d'un DSML doit éviter de définir un langage trop générique ou trop spécifique. Nous avons choisi une grande généricité pour la

partie du DSML concernant le *blackboard* mais avons également introduit dans notre DSML une spécificité forte des modèles grande culture : le modèle phénologique.

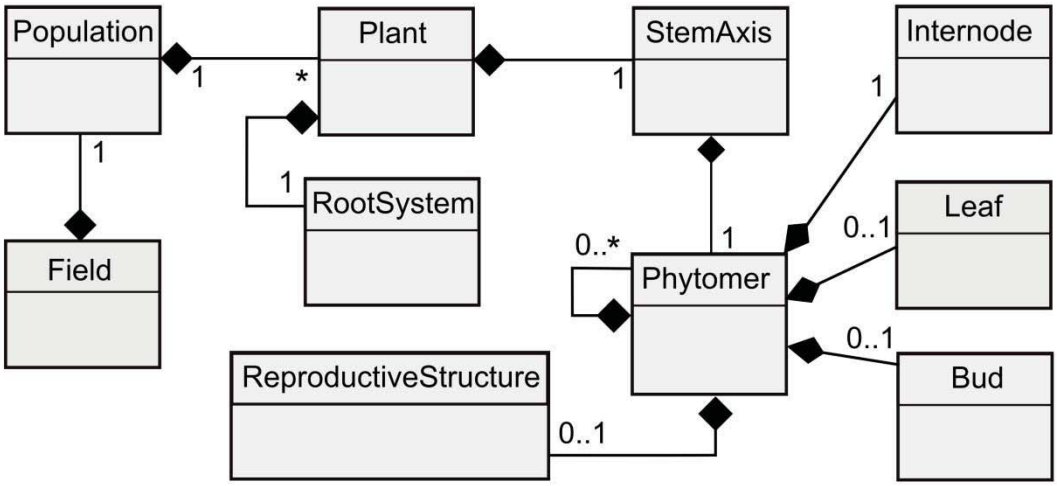


Figure 3.2.5 : Représentation du système plante-sol telle que proposée dans (Barbier *et al.* 2011)

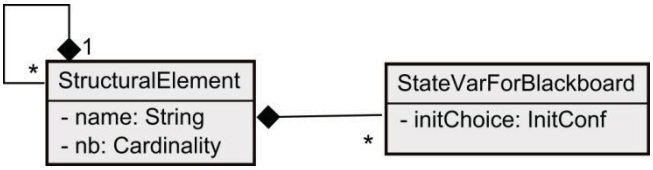


Figure 3.2.6 : Portion de C3M permettant de définir le système plante-sol

2.4- Le modèle phénologique

L'étude du domaine de la modélisation grande culture nous a confirmé l'omniprésence de la représentation du développement de la plante à l'aide d'un modèle phénologique. Celui-ci se traduit par une succession linéaire de stades phénologiques dont les transitions sont assurées par le passage d'une valeur seuil. Généralement, cette valeur est comparée au cumul d'une fonction de la température moyenne journalière. En somme, le modèle phénologique peut être vu comme une machine à état très simple. Etant donné le formalisme retenu pour les modèles exécutables, la définition d'un modèle phénologique donné serait laborieuse. Sachant l'omniprésence de celui-ci, il est apparu préférable d'intégrer au sein de C3M une construction qui lui soit dédiée : *PhenologicalModel* (Figure 3.2.7). Celle-ci permet de spécifier les différents stades phénologiques (*PhenologicalStage*), leur ordre d'apparition ainsi que la fonction (contenue par le *StageAdvancer*) dont la valeur de retour doit être cumulée et comparée avec la valeur seuil du stade phénologique.

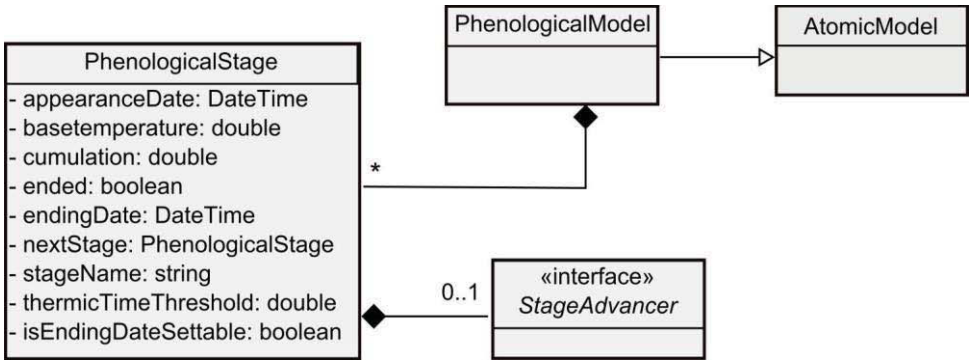


Figure 3.2.7 : Une spécificité le modèle phénologique dans C3M

2.5- Le simulateur

Pour compléter le métamodèle C3M, un concept manque encore pour fédérer les différents processus, la représentation du système plante-sol et les différents fournisseurs de données externes. Nous avons défini ce concept par le terme *Simulator* (cf. figure 3.2.8), bien qu'il ne s'agisse pas d'un simple simulateur. En effet, il comporte bien les informations nécessaires et suffisantes à l'exécution de la simulation, sous réserve de la fourniture des données externes et l'initialisation des différents paramètres, mais il correspond aussi à la seule structure contenant le modèle grande culture complet (modèle composite et blackboard).

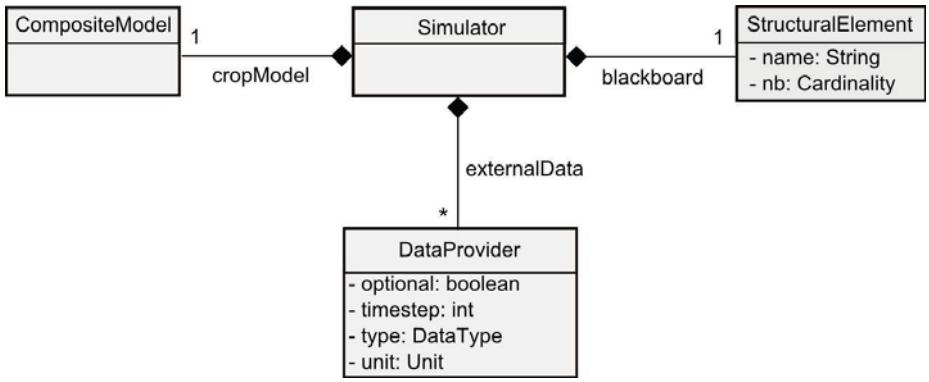


Figure 3.2.8 : Un concept fédérant les éléments constitutifs du modèle grande culture : le simulateur

Cette partie s'est attachée à montrer les concepts constituant le métamodèle C3M, en justifiant l'intégration de différentes spécificités comme les modèles de prétraitement ou le modèle phénologique. La conception de C3M s'est faite de manière itérative en parallèle de la définition du cadriciel destiné à être le support de la génération de code de notre fabrique de modèles agronomiques, CMF, ce cadriciel est présenté dans la partie suivante. Enfin, le

métamodèle C3M constitue la syntaxe abstraite de notre DSML, la dernière partie de ce chapitre en présente la syntaxe concrète souhaitée ainsi que ses modalités de mise en œuvre.

3- Le cadriciel support de la génération automatique de code

La conception du cadriciel a été conduite suivant une idée directrice : faciliter l'intégration d'un modèle et de son simulateur dans un applicatif, en limitant les dépendances entre le simulateur et le reste de l'applicatif. De plus, l'implémentation du couple simulateur/modèle basée sur ce cadriciel doit être lisible pour faciliter les opérations de maintenance. Enfin, les choix de conception ne doivent pas être faits au détriment de la performance d'exécution. Au cours de cette partie, nous présentons successivement l'interfaçage entre le simulateur et l'applicatif, la gestion du temps et de la séquence d'exécution et en dernière partie la gestion des données au cours de la simulation. Nous ne présentons pas uniquement le cadriciel mais également des éléments concrets permettant de comprendre sa mise en œuvre pour une l'implémentation d'un modèle. Quant aux différents concepts de modèle présents dans le cadriciel, ils ne sont qu'évoqués, en effet, ils correspondent exactement aux mêmes concepts que ceux présents dans le métamodèle C3M.

3.1- Définition de l'interface entre le simulateur et l'applicatif

Le terme applicatif est utilisé à dessein dans l'expression « intégration du simulateur dans un applicatif ». En effet, cette intégration peut être menée suivant différents objectifs comme par exemple :

- Mise en place d'un programme destiné à la comparaison de résultats de simulations ;
- Intégration à une plateforme d'analyse de sensibilité ;
- Intégration à un Outil d'Aide à la Décision (OAD).

Les deux premiers cas d'utilisation permettent de valider le modèle et d'en évaluer les performances, ce type d'utilisation conduit à utiliser de manière quasi exhaustive les informations produites au cours de la simulation. Alors que dans le dernier cas, correspondant à l'exploitation industrielle pour laquelle le modèle a été conçu, le nombre d'informations utilisées est beaucoup plus restreint. La rigueur scientifique suppose que l'implémentation du modèle validée soit à l'identique de celle utilisée dans l'OAD, il n'est donc pas possible de changer l'implémentation du couple simulateur/modèle afin de produire plus ou moins d'informations suivant le cas d'utilisation. Compte tenu de ces éléments, soit le couple

simulateur/modèle fournit de manière systématique l'intégralité des informations produites au cours de la simulation soit il permet de spécifier au lancement de la simulation les informations d'intérêt dans le contexte de simulation. La première solution obligerait la personne en charge de l'intégration dans un contexte donné à filtrer parmi cet ensemble d'informations celles qui lui sont nécessaires, de plus, elle obligerait à garder en mémoire des informations sans intérêt dans ce contexte. La deuxième solution, quant à elle, facilite l'intégration en permettant de spécifier les informations attendues, sans impact inutile sur la mémoire. Il a semblé donc plus opportun de retenir cette deuxième possibilité.

Dans ces conditions, un ensemble d'informations sont nécessaires pour l'utilisation du simulateur :

- Les valeurs des paramètres présents dans le modèle ;
- Le cas échéant, les valeurs initiales des variables d'état ;
- Les dates de début et de fin de simulation ;
- Les données nécessaires à la simulation (p.ex. : données météorologiques) ;
- Les sorties souhaitées à l'issue de la simulation.

Ces éléments déterminent la conception présentée par la figure 3.3.1. Dans cette figure apparaissent des éléments spécifiques au cadriciel et des classes en faisant usage dans le cadre du projet vigne nommé « Dispeau ». Les classes spécifiques à ce projet comportent son nom dans leur dénomination. Il existe à ce niveau des adhérences fortes entre le cadriciel et l'implémentation du simulateur Dispeau. Une conception plus générique serait possible, cependant elle pourrait limiter la lisibilité du code ainsi que sa compréhension lors d'opérations de maintenance. Cette moindre généricité se traduit par un plus grand nombre de lignes de code mais, étant donné que leur génération automatique est un objectif de ce travail, ce travers n'aura pas d'impact sur la productivité.

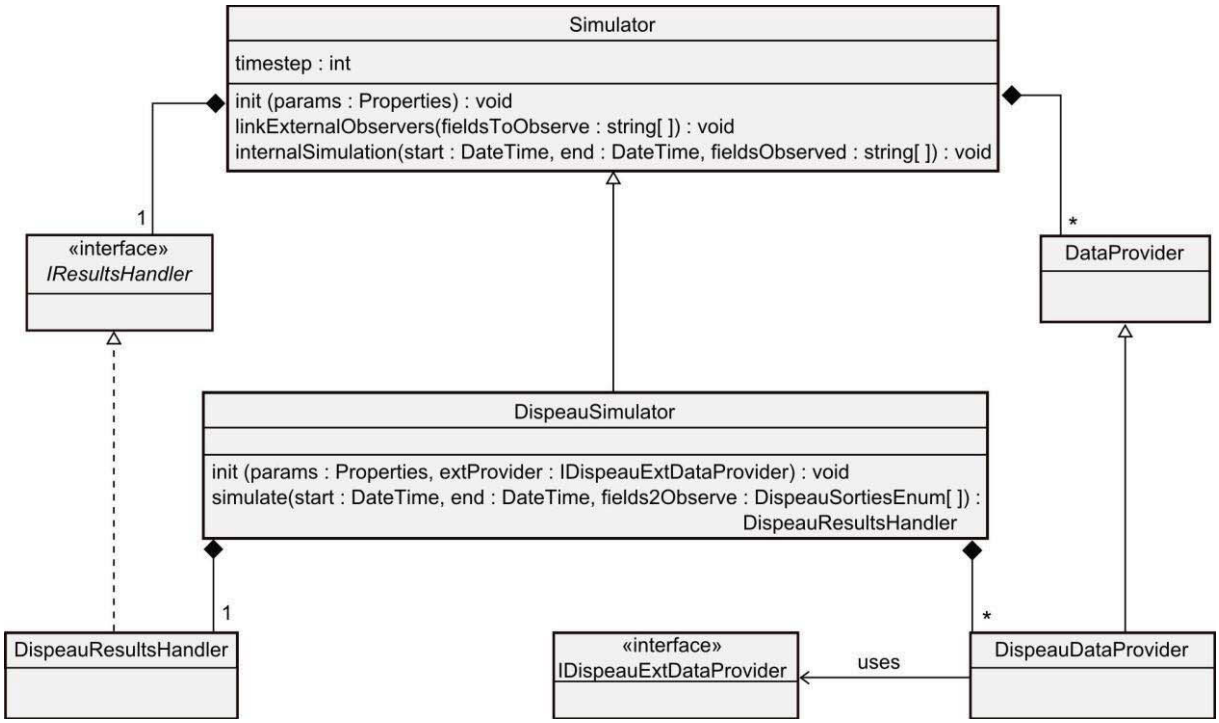


Figure 3.3.1 : Interfaçage du simulateur, diagramme de classes mettant en évidence les classes du cadriciel et d'une implémentation appartenant au projet Dispeau

Les deux classes *Simulator* et *DispeauSimulator* sont les éléments centraux pour l'intégration. La méthode *init()* permet de définir via le paramètre *params* les valeurs de l'ensemble des paramètres et variables d'état nécessaires au bon déroulement de la simulation. Le deuxième paramètre de cette méthode correspond à l'interface qui permet de relier de manière effective les différents *DataProvider* avec une source de données externes : *IDispeauExtDataProvider*. Cette interface permet de gérer plusieurs sources de données (p.ex. : base de données, fichiers, services web) en toute transparence pour les *DataProvider*. La seule règle imposée pour la transmission de données externes est que, quelle que soit la décomposition temporelle de ces données (p.ex. horaire ou journalière), les informations sont transmises par « bloc » correspondant à une journée de données. Ceci correspond à la pratique du domaine.

Les dates de début et de fin de la simulation sont transmises à l'appel de la méthode *simulate* ainsi que les informations souhaitées à l'issue de la simulation. Ces informations sont définies à l'aide d'une énumération listant l'ensemble des informations accessibles. En fonction de la demande, ces informations sont conservées par le *ResultsHandler* retourné par la méthode *simulate*. Cette fonctionnalité repose sur l'exploitation du patron de conception « observateur » (Gamma *et al.* 1995), son application est exposée plus en détail par la suite dans la partie dédiée à la transmission des données (cf. 3.3 de ce chapitre), auparavant, nous

présentons la partie du cadriciel dédiée à la gestion du temps et de la séquence d'exécution au cours de la simulation.

3.2- Ecoulement du temps et séquence d'exécution

Comme écrit précédemment, au sein d'un modèle grande culture peuvent coexister des processus définis à différents pas de temps. Certains algorithmes, identifiés dans ce type de modèles, étaient construits par l'adjonction de boucles internes : dans le simulateur se trouvait une boucle à pas de temps journalier, les modèles construits à un pas temps horaire possédant alors leur propre boucle interne de vingt-quatre heures. Ce traitement ne paraît pas optimal, il a semblé intéressant de modifier l'approche adoptée en définissant deux entités au niveau du cadriciel : une horloge (*Clockwork*) et un workflow permettant de gérer la séquence d'exécution (Figure 3.3.2). Ces deux entités sont initialisées et gérées par le simulateur.

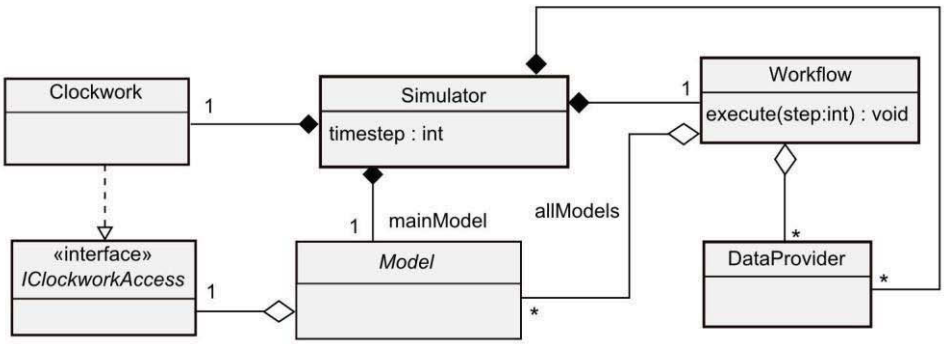


Figure 3.3.2 : Diagramme de classes décrivant la partie du cadriciel dédiée à la gestion du temps et de la séquence d'exécution.

L'horloge est garante de l'écoulement du temps au cours de la simulation. Le simulateur contrôle son avancement par pas de temps. Il initialise l'horloge avec le pas de temps effectif (de façon classique, il s'agit du plus petit pas de temps présent dans les processus). Au lancement de la simulation, le simulateur transmet à l'horloge les dates de début et de fin de simulation. Ceci permet à l'horloge de déterminer le nombre d'itérations à effectuer lors de la simulation. L'horloge implémente une interface *IClockworkAccess* sur laquelle tous les modèles possèdent une référence. Cette interface offre aux modèles la consultation de différentes représentations du temps (p.ex. jours juliens, heure de la journée) ainsi que des fonctions de manipulation du temps (p.ex. comparaison de dates). Ce choix de représentation permet de fournir un ensemble de fonctions utiles aux modèles et centralisées dans une seule classe. L'interface n'offre que la consultation de représentations du temps, le contrôle de son écoulement est assuré par le simulateur.

Le workflow gère la bonne exécution des différentes composantes du modèle en fonction du pas de temps de simulation, de plus il actionne au début de chaque journée la mise à jour des données contenues par les différents dataproviders. Sa méthode d'exécution (*execute*) repose sur l'exploitation de la fonction mathématique modulo appliquée au numéro du pas de temps actuel (paramètre *step* de la méthode), celle-ci permet d'identifier la séquence à exécuter. Cette méthode est mise à plat, c'est-à-dire que l'ensemble des tests sur la fonction modulo et les modèles à exécuter en conséquence sont écrits. Il s'avérerait sans doute un peu fastidieux d'écrire ce code à la main mais dans le cas présent il est destiné à être généré, cas de figure qui nous rapproche des fonctionnalités offertes par les techniques dites d'*inlining*. De plus, ce code permet une plus grande lisibilité lors d'opérations de maintenance.

3.3- Transmission des informations

En ce qui concerne la gestion des données transmises au cours de la simulation, la figure 3.3.3 présente une version allégée du diagramme de classes permettant de comprendre l'implémentation réalisée pour le cadriciel. Tout comme pour le métamodèle, nous avons considéré qu'une information était émise par une source et pouvait être exploitée par un puits. Ceci se traduit au niveau conceptuel par la présence des deux interfaces : *IDataSink* et *IDataSource*. Le rôle de source de données est rempli pour les sorties (*Output*) et les fournisseurs de données (*DataProvider*). Tandis que les entrées jouent le rôle de puits (*Input*). Tout comme dans le métamodèle, les adaptateurs sont à la fois source et puits de données.

La notion d'information, transmise par une source et utilisée par un ou plusieurs puits, nous a amené à considérer qu'une information est un concept partagé entre sources et puits, les unes chargées de la mettre à jour et les autres permettant de la lire. Ceci amène à la classe mère commune à toutes les sources et tous les puits de données *Data<T>* basée sur l'utilisation des generics de Java. Chacune de ces classes contient une référence vers une donnée de type *T*. La figure 3.3.3 explicite un cas pour une sortie de type réelle *RealOutput*, celle-ci est composée d'une classe *Real*. Cette dernière est un *wrapper* (en anglais) qui encapsule une valeur de type primitif *double*. Tout puits exploitant une source possède en réalité une référence vers le wrapper de cette dernière et ne peut accéder à la valeur qu'en lecture. La transmission de la référence du wrapper approprié est réalisée à l'instanciation du simulateur par une classe spécifique *Linker* (non représentée).

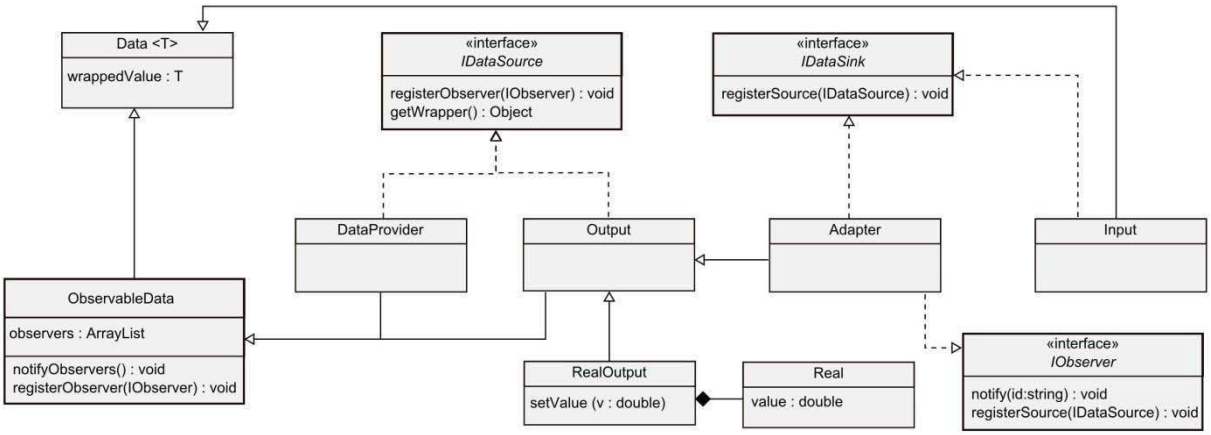


Figure 3.3.3 : Diagramme de classes simplifié de la partie du cadriciel dédiée aux données

Ce choix de conception se rapproche du dataflow tel qu'il est mis en place dans OpenAlea (cf. Chapitre 2) avec comme différence qu'une donnée produite à un pas de temps de la simulation ne peut aucunement être modifiée. Nous voyons plusieurs avantages à ce choix :

- Il limite les adhérences entre les différents modèles. Une entrée exploite l'information pour laquelle elle est définie sans avoir aucun lien avec la source qui l'a produite ;
- Elle permet de garder des méthodes d'exécution simple pour les modèles (*simulateOne*) sans paramétrage. Le workflow peut donc faire appel à ces méthodes sans avoir à gérer la transmission des informations d'un modèle à un autre ;
- De manière plus accessoire, la majorité des instances nécessaires à l'exécution de la simulation sont créées dès l'instanciation du simulateur. Ceci limite le nombre d'instanciation par rapport à d'autres choix d'implémentation. Couplé à l'utilisation de type primitif ceci devrait conduire à des performances supérieures en termes de temps de calcul.

Toutes les sources de données ont une classe mère commune *ObservableData*, celle-ci permet d'enregistrer des observateurs suivant le patron de conception du même nom (Gamma *et al.* 1995), les variables d'état des modèles ainsi que celle contenues dans la représentation du système plante-sol héritent également de cette classe. La mise à jour de la valeur contenue par une instance de ces classes déclenche une notification aux observateurs enregistrés, suivant la qualité de l'observateur, différents traitements sont réalisés. En effet, deux types d'observateurs existent. Le plus courant est l'observateur mis en place à la demande de la simulation, comme écrit ci-avant, le lancement de la simulation s'effectue en spécifiant les

champs constituant les sorties de la simulation. Le *ResultsHandler* enregistre différents observateurs en conséquence. Le deuxième cas de figure est inhérent à certains adaptateurs. En effet, pour les adaptateurs responsables de l'agrégation de plusieurs informations, il est nécessaire que leur soit signalée la mise à jour d'une source afin de pouvoir assurer certains traitements et *in fine* de mettre à jour le conteneur permettant d'alimenter les puits auxquels ils sont reliés. Ce cas de figure est valable à la fois pour les adaptateurs gérant plusieurs sources (l'agrégateur du métamodèle) ou une seule source pour l'adaptateur temporel. Ce dernier cas est explicité par un exemple concret dans la figure 3.3.4.

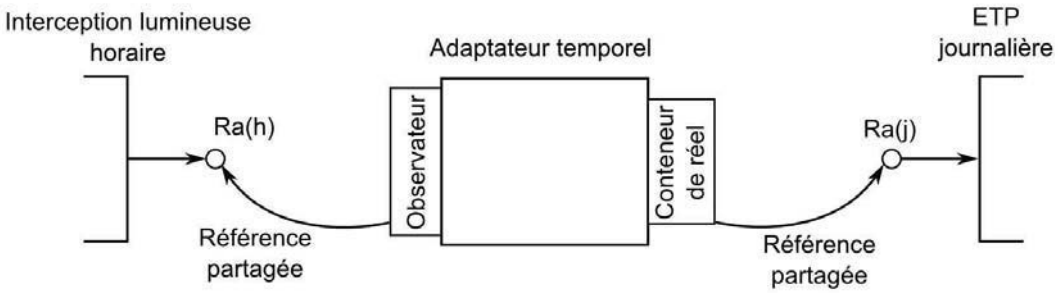


Figure 3.3.4 : Schéma explicitant la modalité de fonctionnement d'un adaptateur assurant la liaison entre une source fournissant la radiation absorbée à un pas de temps horaire Ra(h) et un puits exploitation la radiation absorbée à un pas de temps journalier Ra(j)

Cette figure représente un cas d'adaptation issu du modèle de croissance de la vigne. Dans ce modèle, un processus d'interception lumineuse (partie gauche de la figure) calcule la radiation absorbée à un pas de temps horaire. Dans la séquence d'exécution, le processus suivant détermine, à l'aide de la radiation absorbée, l'évapotranspiration potentielle (ETP) à un pas de temps journalier (partie droite de la figure). Un adaptateur est nécessaire entre la production horaire de l'information radiation absorbée et son utilisation à un pas de temps journalier, son rôle est d'effectuer un cumul des vingt-quatre valeurs horaires. Cet adaptateur joue le rôle de source de données pour l'entrée radiation absorbée du processus calculant l'ETP, conformément à ce que nous avons exposé ci-avant, l'adaptateur contient un conteneur de type réel dont la référence est partagée avec cette entrée. Pour pouvoir assurer le suivi des valeurs produites par la sortie du processus d'interception lumineuse, l'adaptateur dispose d'un observateur dont la référence est partagée avec cette sortie. Chaque notification de mise à jour déclenche un cumul dans une variable intermédiaire et le décompte du nombre de notifications reçues. Après vingt-quatre notifications, la valeur encapsulée par le wrapper est mise à jour, le cumul et le décompte des notifications sont réinitialisés. La réalisation du

cumul des valeurs par une valeur intermédiaire est indispensable, il aurait été impossible de l'effectuer directement via la valeur encapsulée. En effet, s'agissant d'une source de données, celle-ci est observable, son utilisation directe pour le cumul aurait déclenché des notifications intermédiaires pour des valeurs ne correspondant pas à l'information attendue pour cette source.

Dans cette partie, ont été présentés les éléments majeurs qui ont guidé la conception du cadriciel ainsi que les choix d'architecture retenus. Le cadriciel a servi de support à la réingénierie des modèles blé et vigne, conformément à l'approche dirigée par les modèles exposée au chapitre 2 de ce mémoire. Cette réingénierie a entrainé quelques évolutions mineures du cadriciel sans pour autant remettre en question son architecture. L'objectif pour ce cadriciel est de limiter la quantité de code à générer dans le cadre de la fabrique de modèles. Avant de pouvoir procéder à cette génération, il est nécessaire de concevoir l'interface de conception de modèles agronomiques. Celle-ci s'appuie sur l'utilisation d'un DSML (Domain-Specific Modelling Language) dont la syntaxe concrète est spécifiée dans la section suivante.

4- Syntaxe concrète du DSML et description des éditeurs attendus

Comme expliqué au chapitre 2, la syntaxe concrète d'un DSML correspond aux éléments graphiques et/ou textuels utilisés par le modélisateur pour manipuler les concepts présents dans le métamodèle et ainsi définir un modèle conforme à ce dernier. Dans le cas de CMF, l'objectif ultime est de fournir une syntaxe concrète hybride : une partie graphique pour la définition de la hiérarchie de modèles, les flux d'information et la séquence d'exécution, puis une partie textuelle pour la définition des algorithmes représentant les processus biophysiques. La partie textuelle n'a pu être abordée au cours de la thèse, les lignes directrices pour sa définition et les fonctionnalités à lui associer sont évoquées dans la dernière partie de ce mémoire (perspectives au chapitre 4). Les lignes qui suivent ne présentent donc que les aspects graphiques de notre DSML. Elles visent à donner les représentations des différents concepts retenus pour le DSML ainsi qu'à préciser leur modalité d'agencement les unes par rapport aux autres. De plus, sont explicités les différents éditeurs imaginés pour la conception de modèles grande culture ainsi que certaines fonctionnalités devant faciliter l'édition.

4.1- Les représentations graphiques du DSML et les règles de disposition

D'une manière générale, il est difficile de s'affranchir du formalisme boîte-ligne lors de la définition d'une syntaxe graphique. Les différentes boîtes permettent de représenter les concepts du domaine et les lignes la manière dont ces concepts sont associés. Dans le cadre de CMF, les éléments de la syntaxe graphique peuvent se décomposer en trois catégories : les modèles, les sources et puits de données, les éléments définissant la séquence d'exécution.

4.1.1- Représentation des modèles

A partir du métamodèle C3M, quatre types de modèles nécessitant une représentation graphique sont identifiés (cf. Tableau 3.1). Nous avons souhaité mettre en place pour les modèles des visuels simples, rendant à la fois compte de leur appartenance à une même catégorie tout en permettant d'identifier rapidement la spécificité de chacun d'entre eux. Certaines informations spécifiques au modèle n'ont pas de représentation visuelle, elles sont destinées à être saisies dans une fenêtre de propriétés. Il s'agit du pas de temps, de la description du modèle, ainsi que des paramètres et variables d'état pour les modèles exécutables autres que le modèle phénologique. En effet bien que jouant le rôle de source de données pour la simulation, les variables d'état d'un modèle exécutable ne sont pas destinées à transmettre des informations à d'autres éléments du modèle grande culture. Dès lors, les faire apparaître dans la représentation graphique ne ferait qu'alourdir la conception et n'améliorerait pas la compréhension de la structuration générale du modèle. En effet, l'apport de la représentation graphique est d'offrir une bonne appréhension de la hiérarchisation du modèle grande culture, de la transmission d'information et de sa séquence d'exécution.

Modèle composite		Le modèle composite a comme seul objectif de permettre la décomposition du modèle grande culture en sous-ensembles logiques
Modèle atomique		Un modèle atomique contient la logique mathématique du processus biophysique qu'il représente
Modèle de prétraitement		Modèle exécutable comme le modèle atomique, il intervient au début de la période du pas de temps pour lequel il a été défini. (cf. 2.1 de ce chapitre)
Modèle phénologique		Un modèle atomique spécifique, il permet de définir les stades phénologiques et leurs règles d'avancement respectives

Tableau 3.1 : La représentation des modèles dans la syntaxe concrète du DSML

4.1.2- Représentation des sources et des puits de données

Sept éléments de la syntaxe graphique permettent de représenter des concepts de source et/ou de puits de données (cf. Tableau 3.1). Un huitième élément, un lien en pointillés, correspond à la représentation du flux d'information entre une source et un puits de données. Les entrées et sorties présentent une particularité dans la syntaxe définie, elles n'ont pas d'existence autonomes et sont obligatoirement accolées au modèle exécutable auquel elles appartiennent. Aucune représentation graphique n'a été intégrée pour le *blackboard* qui possède un éditeur spécifique (cf. partie 4.2.3 de ce chapitre).

Entrée		Entrée d'un modèle exécutable, la destination de la flèche est accolée au modèle exécutable
Sortie		Sortie d'un modèle exécutable, la flèche a pour origine le modèle exécutable produisant cette information
Adaptateur temporel		Ce type d'adaptateur entre en jeu lorsqu'une entrée exploite une donnée à un rythme différent de celui auquel cette donnée est produite (*p.ex.* entrée d'un modèle journalier liée à une sortie d'un modèle horaire)
Convertisseur		Intervient lorsque la source de donnée produit une information dont les unités sont différentes de celles attendues par une entrée
Agrégateur		Lorsque plusieurs sources d'information doivent être agrégées en une seule (*p.ex.* transpiration et évaporation réelles sont transmises à une entrée représentant l'évapotranspiration réelle)
Adaptateur Conditionnel		Lorsqu'une entrée est reliée à deux sources de données et qu'une seule doit être prise en compte lors de l'exécution (*p.ex.* l'ETP peut être fournie en entrée du modèle global si ce n'est le cas un modèle permet de calculer l'ETP qui doit être prise en compte à la place de l'information fournie par le dataprovider)
DataProvider		Source d'informations externes nécessaires au déroulement de la simulation

Tableau 3.2 : Les puits et sources de données dans la syntaxe concrète du DSML

4.1.3- Les éléments intervenant dans la séquence d'exécution

Concernant la séquence d'exécution, seuls trois éléments graphiques sont nécessaires (cf. Tableau 3.3). Deux flèches permettent de définir pour l'une le premier élément de la séquence d'exécution et pour l'autre le successeur d'un modèle dans cette séquence. Le troisième élément correspond à la possibilité de définir sous une certaine condition une séquence d'exécution alternative. Comme annoncé ci-avant, le concept permettant de définir une séquence conditionnelle est analogue à un modèle dans la syntaxe abstraite. Cependant, il n'intervient sémantiquement que dans la définition de la séquence et non dans la transmission d'information comme pour les autres modèles.

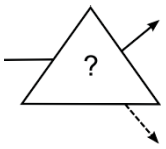
Premier modèle		Désigne le premier modèle à intervenir dans la séquence d'exécution
Modèle suivant		Indique l'ordre de succession des modèles dans la séquence d'exécution
Séquence conditionnelle		Permet de changer la séquence d'exécution en fonction de la condition testée. La flèche pleine (respectivement en pointillés) pointe vers le modèle suivant dans le flux si la condition est vraie (respectivement fausse)

Tableau 3.3 : Eléments de la syntaxe graphique intervenant dans la définition de la séquence d'exécution

4.2- Les différents éditeurs et fonctionnalités associées

Pour pouvoir manipuler les éléments de syntaxe graphique qui viennent d'être exposés et procéder à la conception d'un modèle grande culture, quatre éditeurs spécifiques sont nécessaires. Ces éditeurs ne doivent pas seulement permettre de concevoir un modèle grande culture conforme au métamodèle C3M mais également offrir des possibilités de construction automatique en adoptant un comportement aussi « intelligent » que possible grâces aux connaissances présentes dans le métamodèle.

4.2.1- Le point d'entrée : l'éditeur du simulateur

Le premier éditeur constitue le point de départ de la conception d'un nouveau modèle agronomique. Il permet de définir le contenu du simulateur. La figure 3.4.1 représente l'éditeur du simulateur tel qu'il a été envisagé. Le nombre de concepts qu'il est possible d'intégrer dans cet éditeur est restreint. En effet, il ne permet d'ajouter que des *dataproviders* et présente deux éléments immuables : le modèle principal et le *blackboard*. Ces deux éléments permettent d'accéder à leurs éditeurs spécifiques : l'éditeur de modèle composite pour le premier (cf. ci-après 4.2.2) et l'éditeur du *blackboard* (cf. ci-après 4.2.3). Un système d'infobulles donne un aperçu pour les *dataproviders* de la donnée qu'il représente, de la fréquence à laquelle elle est fournie et de l'identification des puits du modèle grande culture exploitant cette information.

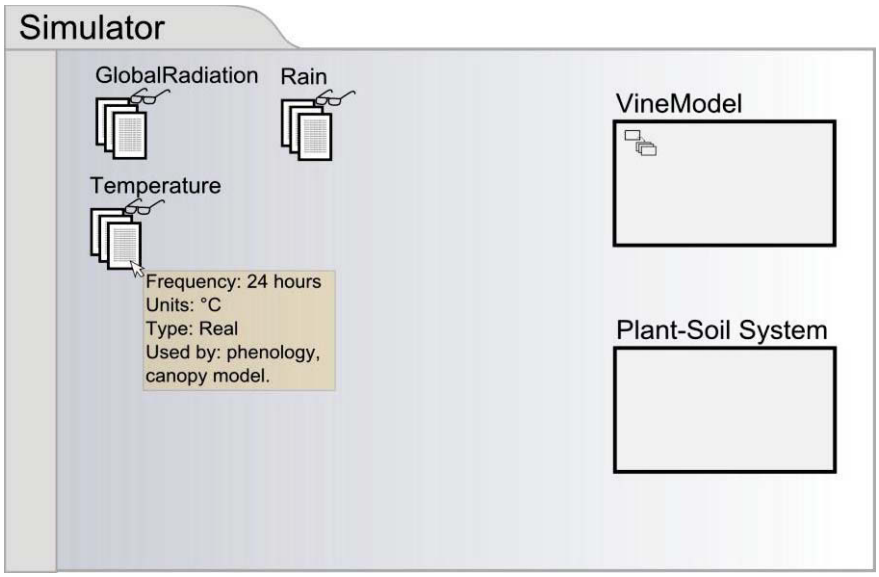


Figure 3.4.1 : Schéma de l'éditeur du simulateur pour CMF

4.2.2- L'éditeur de modèles composites

L'éditeur de modèles composites est le plus complexe des éditeurs à réaliser. En effet, mis à part les *dataproviders*, l'ensemble des éléments de la syntaxe graphique peuvent y intervenir. C'est aussi l'éditeur qui a vocation à être utilisé le plus couramment par les modélisateurs. Depuis cet éditeur, il est possible d'ouvrir :

- d'autres éditeurs de modèles composites et ainsi de parcourir la hiérarchie de modèles ;
- l'éditeur de modèles phénologiques (cf. 4.2.4).

La figure 3.4.2 fournit une représentation de cet éditeur ainsi que différentes vues qui lui sont associées. En effet, compte tenu du rôle central de cet éditeur, il paraît important de fournir des vues synthétiques pour permettre au modélisateur d'obtenir aisément des informations sur le modèle grande culture en cours de conception. La figure présente deux vues possibles, un résumé du workflow de simulation en fonction des différents pas de temps (Simulation flow, en bas) et une présentation arborescente de la hiérarchie des modèles avec leurs entrées et leurs sorties (ModelTree, à droite).

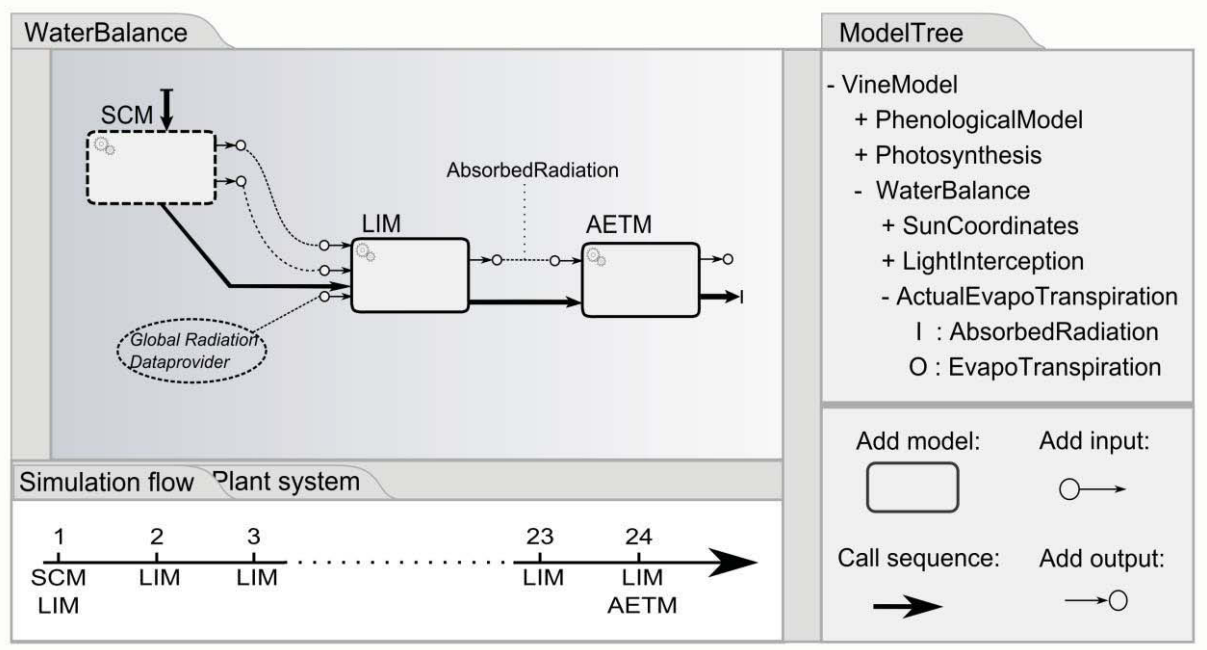


Figure 3.4.2 : Editeur d'un modèle composite

Certaines fonctionnalités dans cet éditeur devraient faciliter la conception du modèle grande culture. Elles peuvent se rapprocher des fonctionnalités d'auto-complétion offertes par les environnements de développement classiques. Notamment, la mise en place d'une liaison entre une source et un puits de données déclenche une comparaison entre ces deux concepts :

- par rapport aux unités et dimensions qu'elles déclarent ;
- par rapport au rythme d'émission de la source et au rythme de lecture du puits ;
- par rapport à l'existence d'une autre source alimentant ce puits.

Ces contrôles permettent de créer les adaptateurs nécessaires à la liaison de données entre cette source et ce puits. De plus, l'utilisation de glisser-déposer d'une sortie vers un modèle exécutable crée une nouvelle entrée sur ce modèle, l'associe à cette sortie, le cas échéant, par l'intermédiaire d'un adaptateur. Pour l'utilisateur, ce type de fonctionnalités devrait faciliter la conception de modèles ainsi qu'améliorer la productivité.

4.2.3- Editeur pour le *blackboard*

Pour la conceptualisation du *blackboard* dans le métamodèle C3M, deux types de concepts interviennent (cf. ci-avant figure 3.2.6) : les variables d'état et les éléments structurels. Une variable d'état correspond à une quantité mesurable comme une hauteur ou une surface. Un élément structurel représente une subdivision hiérarchisée de la structure de données, c'est par

cette construction qu'il va être possible de donner, si nécessaire, une dynamique à la structure de données. Ces éléments structurels correspondent à la notion d'organes pour la plante ou à la notion de couches pour un sol. Un élément structurel peut contenir des variables d'état ou d'autres éléments structurels : par exemple un phytomère porte une feuille et éventuellement une ramification. Dans la structure de données, le sol et la plante sont deux éléments structurels.

Etant donné le peu de concepts nécessaires à la représentation du *blackboard* et leur agencement, il nous est apparu peu opportun de proposer une syntaxe graphique pour cet éditeur et plus approprié de le présenter sous la forme d'une arborescence. Un menu contextuel sur les éléments structurels offre la possibilité d'ajouter une nouvelle variable d'état ou un nouvel élément structurel. Les figures 3.4.3 et 3.4.4 présentent cet éditeur avec deux structures de données différentes. La figure 3.4.3 présente les options disponibles pour la création d'une variable d'état. Trois options sont disponibles pour le typage d'une variable d'état : réel, entier ou énuméré. Le champ « Valeur initiale » permet de sélectionner la manière dont la variable d'état est initialisée :

- Paramètres : la valeur initiale est transmise, à l'instar des paramètres, à l'appel du simulateur. Cas d'une variable d'état dépendant du contexte de simulation comme par exemple la quantité d'eau disponible dans le sol.
- Création : au moment où la structure qui contient cette variable d'état est créée, la valeur initiale est fournie.
- Fixe : la valeur initiale sera toujours la même quel que soit le contexte. Par exemple, la valeur initiale de la surface foliaire peut être considérée égale à zéro quel que soit le cas de figure.

Le champ « Valeur par défaut » permet de renseigner la valeur initiale fixe. Si la valeur initiale correspond à « Paramètres » ou « Création », la valeur par défaut ne doit être renseignée que si l'utilisateur souhaite instaurer un comportement par défaut en l'absence de fourniture de l'information requise dans les paramètres ou à la création de la variable.

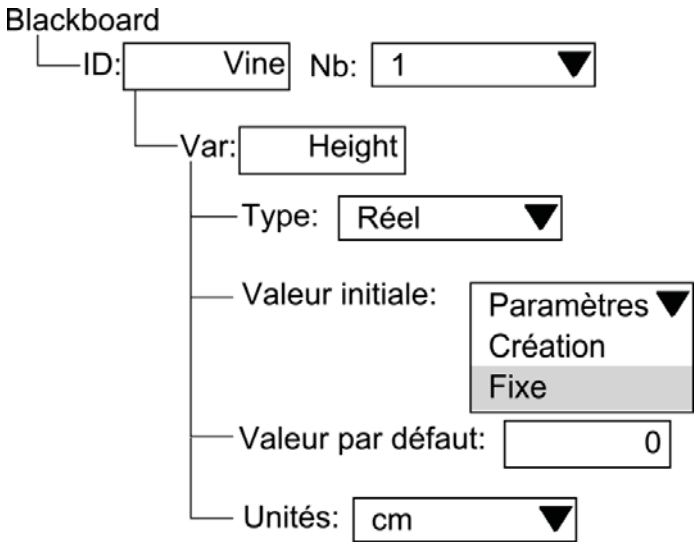


Figure 3.4.3 : Options de déclaration d'une variable d'état

La figure 3.4.4 se focalise sur une arborescence d'éléments structurels. Peu d'informations sont nécessaires pour décrire ces éléments, chaque élément est caractérisé par son identifiant ainsi qu'une cardinalité. La cardinalité *1..n* pour le phytomère signifie que son élément parent - la tige - comporte au minimum un phytomère, présent dès l'initialisation, et d'autres sont ajoutés en cours de simulation. Chaque phytomère porte une et une seule feuille, en cours de simulation une ramification peut apparaître sur ce phytomère. Cette ramification est déclarée en employant de nouveau l'identifiant *Tige* avec une cardinalité *0..1*. La reprise de cet identifiant permet de répéter, lors de la simulation, la représentation Tige / Phytomère / Feuille lorsqu'une ramification est créée à partir d'un phytomère.

L'utilisation des cardinalités pour les éléments structurels présente l'avantage de permettre la bonne initialisation du système de données plante-sol au lancement de la simulation. De plus, les cardinalités précisent les éléments qui seront créés en cours de simulation. Grâce à elles, il est donc possible de déterminer les méthodes de création d'éléments nécessaires au modélisateur. De plus, en fonction des conditions d'initialisation des variables d'état des éléments créés par ces méthodes, il est possible de déduire les paramètres nécessaires à ces méthodes pour aboutir à l'initialisation des variables d'état.

Le dernier éditeur souhaité pour CMF, correspond à l'éditeur dédié au modèle phénologique. Cet éditeur est simple, il permet de définir la succession des stades phénologiques dans le modèle ainsi que leurs règles d'avancement respectives. La syntaxe graphique pour cet éditeur est donc dédiée à la représentation de stades et à l'utilisation de liens orientés pour définir la succession de stades. Les descripteurs de chaque stade phénologique, comme son nom ou sa

règle d'avancement, sont mis à jour via une fenêtre de propriété. La syntaxe graphique du DSML et les éditeurs de CMF sont des propositions qui ont servi de base à la réalisation d'un prototype de fabrique de modèles grande culture. Elles seront vraisemblablement amenées à évoluer par la suite.

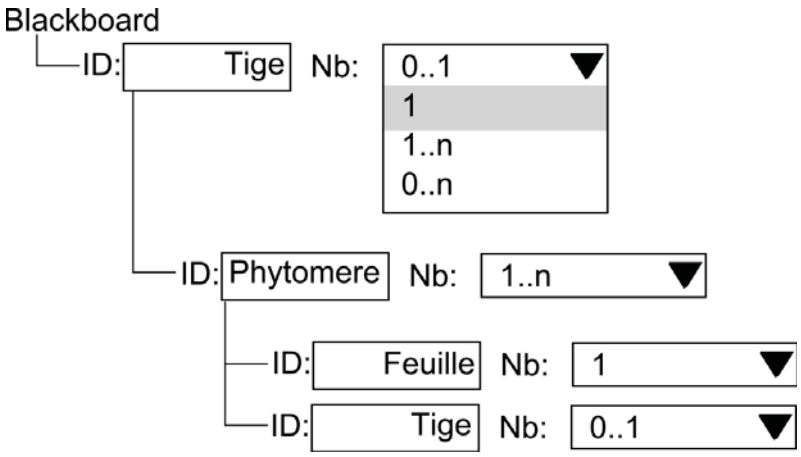


Figure 3.4.4 : Edition du *blackboard*, cas concret pour la mise en évidence des éléments structurels

5- Conclusion

Conformément à la démarche d'Ingénierie Dirigée par les Modèles définie au chapitre 2, nous avons exposé au cours de ce chapitre une analyse du domaine de la modélisation grande culture. Cette analyse s'est appuyée à la fois sur des modèles patrimoniaux de la société ITK et sur une revue la plus complète possible de la littérature relative à la modélisation grande culture. Cette analyse nous a permis d'identifier les concepts caractéristiques de ce domaine de modélisation. Ces concepts définissent la syntaxe abstraite de notre DSML qui a été formalisée sous la forme d'un métamodèle : C3M (*Crop Model Metamodel*). Nous avons également établi un ensemble de formalismes graphiques constituant la syntaxe concrète du DSML. Enfin, la sémantique associé à notre langage trouve sa réalisation dans le cadriciel que nous avons conçu et détaillé ci-avant, ce cadriciel a été conçu et testé sur les modèles de croissance de la vigne et du blé.

En accord avec la définition d'un langage telle que donnée par (Gray *et al.* 2007) (cf. chapitre 2 section 1.2), une fois obtenue les syntaxes abstraite et concrète et la sémantique du langage, il reste alors à définir les applications M_S et M_C donnant, respectivement, la correspondance entre la syntaxe abstraite et la sémantique et la correspondance entre la syntaxe concrète et la

syntaxe abstraite. La définition de ces deux applications est obtenue par l'implémentation effective de notre fabrique de modèles grande culture CMF.

Dans le prochain chapitre, nous présentons l'interface mise en place pour CMF, cette interface avec ses éditeurs correspond à l'application de correspondance entre syntaxe concrète et syntaxe abstraite. Nous explicitons ensuite la solution de génération de code intégrée à CMF. Le code généré s'appuie sur le cadriciel que nous venons de détailler, ainsi la génération de code permet-elle de définir l'application de correspondance entre la syntaxe abstraite et la sémantique de notre langage. Un cas concret d'utilisation du prototype obtenu est présenté en prenant comme exemple le modèle de croissance de la vigne. Enfin, nous discutons les résultats obtenus et abordons les perspectives concernant CMF et l'IDM.

Chapitre 4 – Prototypage de la fabrique de modèles

Au cours du chapitre précédent, nous avons présenté les éléments conceptuels nécessaires au prototypage de CMF (Crop Model Factory). De nombreux environnements étaient envisageables pour développer ce prototype. Cependant, l'évaluation de l'ensemble de leurs possibilités et des fonctionnalités, qui auraient pu être utiles pour le développent de CMF, n'a pas pu être menée dans le temps imparti à la thèse. Compte tenu de nos contraintes et des besoins que nous avons exprimés dans les chapitres précédents, notre choix s'est porté sur l'environnement Eclipse et ses plugins orientés IDM (Ingénierie Dirigée par les Modèles). Ce choix nous a semblé être le compromis le plus adapté pour réaliser le prototype. Nous présentons, en première partie, un rapide aperçu d'Eclipse avec une focalisation sur les plugins orientés IDM. Leur mise en œuvre est ensuite abordée pour l'obtention de l'interface graphique de CMF. Cette interface permet la conception visuelle des modèles grande culture, ainsi que la génération automatique de leur code. Un exemple concret, basé sur le modèle de croissance de la vigne, est donné en troisième partie. Enfin, nous discutons la réalisation de CMF avant de livrer les perspectives envisagées. Cette dernière partie met également en évidence des besoins spécifiques quant à l'outillage disponible pour la communauté IDM.

1- Environnement retenu pour le prototype

1.1- La plateforme Eclipse

Plusieurs éléments justifient le choix de la plateforme Eclipse en tant que compromis raisonnable pour concevoir le prototype de CMF. En premier lieu, Eclipse est une plateforme open-source, son utilisation en tant qu'environnement de développement est particulièrement répandue dans la communauté Java. Il s'agit d'autre part de l'environnement retenu par les développeurs d'ITK pour la réalisation des applications orientées aide à la décision. Son architecture, basée sur le concept de plug-in, est orchestrée par la fondation d'Eclipse. Cette décomposition en différents plug-ins facilite la gestion par la fondation de différents projets indépendants. Grâce à elle, il est possible pour les développeurs d'ajouter de nouvelles fonctionnalités à leur environnement en réponse à leurs besoins. L'un des projets d'intérêt pour notre travail est l'Eclipse Modeling Project[24]. Particulièrement actif depuis une dizaine d'années, il vise à doter Eclipse de fonctionnalités orientées IDM. Il est décomposé en un ensemble de sous-projets focalisés sur des aspects spécifiques de l'IDM : métamodélisation, édition graphique de modèles, édition textuelle, transformations, persistance de modèles.

Dans le cadre du travail de thèse, nous nous sommes orientés sur l'utilisation :

- d'EMF (Eclipse Modeling Framework), qui est l'élément de base de la métamodélisation sous Eclipse ;
- de GMF (Graphical Modeling Framework), permettant la définition et la mise en œuvre d'une syntaxe concrète graphique pour un DSML (Domain-Specific Modeling Language) ;
- d'EuGENia, qui a pour objectif de faciliter l'utilisation d'EMF et de GMF ;
- d'Acceleo, pour la génération automatique de code.

1.2- Métamodélisation avec Eclipse : EMF

EMF (Steinberg et al. 2008) est l'un des projets les plus anciens de la fondation Eclipse sur la thématique de l'IDM. Sa principale fonctionnalité - dans notre contexte - est de permettre la définition d'un métamodèle en se basant sur le méta-métamodèle Ecore. De plus, il permet la conception de modèles conformes à ce métamodèle par la génération du code Java d'un éditeur simple sous forme arborescente. EMF fournit également une API de réflexion

[24] http://projects.eclipse.org/projects/modeling

permettant la création et la manipulation de modèles de manière programmatique. EMF est également doté d'une API de validation, celle-ci vérifie la conformité syntaxique des modèles créés. Enfin, EMF est doté d'une API de persistance qui, par défaut, génère un fichier XMI (XML Metadata Interchange). Ce fichier contient l'ensemble des descripteurs correspondant au modèle conçu.

1.3- Syntaxe concrète graphique avec GMF

Le projet GMF (Gronback 2009) est dédié à la définition d'une syntaxe graphique concrète et à la génération automatique d'éditeurs permettant de concevoir des modèles reposant sur cette syntaxe. GMF est une extension du Graphical Editing Framework (GEF). Tout comme GEF, GMF repose sur l'exploitation du patron de conception Modèle-Vue-Contrôleur pour ces éditeurs.

Le principe de fonctionnement de GMF permet d'obtenir plus rapidement des éditeurs riches par rapport à l'utilisation directe de GEF. Ce principe repose sur la définition de trois fichiers pour chaque type d'éditeurs. Un type d'éditeur correspond à un diagramme GMF. A chaque diagramme sont associés les fichiers de :

- définition graphique : permet de définir les éléments syntaxiques disponibles dans le diagramme ;
- définition de la palette : définit les outils accessibles pour l'ajout de nouveaux éléments dans le diagramme ;
- définition des correspondances : permet de faire la liaison entre le métamodèle, la syntaxe concrète et les éléments de la palette.

Après définition, ces trois fichiers sont fusionnés dans un fichier de génération du diagramme d'édition. C'est à partir de ce dernier qu'est généré le code Java correspondant à l'éditeur.

Dans la pratique, nous avons utilisé EuGENia (Kolovos et al. 2010). Cette solution permet, à partir d'un fichier unique décrivant le métamodèle, d'y ajouter des annotations spécifiques et de générer automatiquement les fichiers de définition graphique, de la palette et de correspondance requis par GMF. Cela nous a permis d'obtenir plus rapidement un ensemble d'éditeurs dont le comportement a ensuite été adapté.

1.4- Génération de code avec Acceleo

L'obtention du code Java correspondant aux modèles grande culture conçus à partir de CMF a été réalisée à l'aide Acceleo. Cette solution, développée par la société Obeo, a été offerte à la fondation Eclipse et intégrée au projet M2T (Model To Text) dédié à la génération de code à partir de modèles. Elle se base sur un système de 'templates' appelés modules. Un module principal est défini comme point de départ de la génération de code. Acceleo exploite sous la forme d'un fichier XMI le modèle à transformer ainsi que le métamodèle auquel il se conforme. L'utilisation de modules permet d'obtenir un code particulièrement lisible. Le langage de script utilisé est assez simple d'abord, il est possible de le combiner avec des fonctionnalités plus avancées à l'aide du langage OCL (Object Constraint Language). Ce dernier offre la possibilité d'effectuer des requêtes sur l'ensemble du modèle fourni afin d'obtenir des informations synthétiques. Acceleo gère un cache de requête OCL, évitant ainsi une interrogation supplémentaire du modèle lorsqu'une requête OCL est utilisée plusieurs fois au cours de la génération.

2- Mise en œuvre et difficultés rencontrées

2.1- Obtention des éditeurs

La réalisation de l'interface graphique de la fabrique CMF a été réalisée par un stagiaire élève-ingénieur de deuxième année de l'ISIMA (Flusin 2012) sous la supervision du doctorant. Son travail s'est focalisé sur la mise en œuvre de GMF et d'EuGENia, ainsi que sur l'adaptation du code des éditeurs générés. Le métamodèle a été spécifié dans un fichier emfatic en utilisant les annotations nécessaires à EuGENia pour obtenir les fichiers de configuration GMF. La syntaxe d'emfatic est proche de la syntaxe Java, elle permet de décrire rapidement le métamodèle relativement simple qu'est C3M (Crop Model MetaModel). Un extrait d'un des fichiers emfatic est livré ci-après (cf. code 4.1). Ce fichier a été conçu afin de permettre la génération de l'éditeur spécifique aux modèles composites. Il débute par la définition d'un package qui correspond au métamodèle, l'annotation associée à ce package permet de définir l'URI (Uniform Resource Identifier) du métamodèle. Grâce à cet URI, les différents outils intervenant dans le cycle de modélisation et de transformation de modèles peuvent accéder au métamodèle. Nous présentons le cas pour Acceleo dans la suite de cette partie.

Hormis la déclaration de package, trois classes sont visibles dans notre exemple : *CompositeModel, AlternateSequenceModel, ExecutableModel*. Concernant le modèle composite, deux annotations apparaissent :

- la première, *gmf.diagram*, indique que le diagramme a pour vocation d'éditer le contenu d'un modèle composite. Notons que si l'annotation ne contient pas de paramètres elle ne sera pas prise en compte par EuGENia, d'où l'emploi de la notation *foo=bar* qui n'a aucune signification particulière ;
- la seconde, gmf.node, indique la représentation graphique donnée aux modèles composites contenus par le modèle en cours d'édition.

Le modèle composite contient une référence vers le premier de ses modèles intervenant dans la séquence d'exécution. Cette référence est définie à l'aide du mot-clé **ref** suivi du type de l'élément référencé, de la cardinalité et du nom de l'association. Les deux autres attributs correspondent aux éléments qui composent le modèle : un nombre strictement positif de modèles et de 0 à n adaptateurs.

```
/*Annotation permettant de définir l'uri du métamodèle*/
@namespace(uri="itk.tooling.model.c3m", prefix="c3M")
package c3M;

/*Permet de définir le type élément racine pour le diagramme : ici un modèle composite*/
@gmf.diagram(foo="bar")
@gmf.node(label = "ID", label.placement = "external")
class CompositeModel extends Model {
  ref Model[1] firstInFlow;
  val Model[+] subModels;
  val Adapter[*] adapters;
}

@gmf.node(label = "ID", figure = "rounded", label.placement = "external")
class AlternateSequenceModel extends Model {
  @gmf.link(target.decoration = "arrow", style = "dot", color = "255,255,0")
  ref Model[1] alternateInFlow;
  @gmf.link(target.decoration = "arrow", style = "dot", color = "0,0,0")
  ref DataSource[*] sources;
  attr String[1] rule = "rule";
}

abstract class ExecutableModel extends Model {
  op void executeOneStep();
  @gmf.affixed(foo="bar")
  val Input[*] inputs;
  @gmf.affixed(foo="bar")
  val Output[*] outputs;
  val Parameter[*]#belongsTo parameters;
  val StateVariable[*] stateVariables;
}
```

Code 4.1 : Extrait du fichier emfatic pour la génération de notre éditeur de modèles composites

Comme le soulignent Kolovos et ses coauteurs (Kolovos *et al.* 2010), l'approche offerte par EuGENia présente l'inconvénient de placer au même niveau la syntaxe abstraite et la syntaxe concrète du DSML. Cependant, dans le cadre d'une preuve de concept, il nous est apparu plus important de gagner du temps sur la réalisation que de choisir la solution GMF seule, même si cette dernière est plus formelle et plus correcte du point de vue des concepts sur la séparation des représentations.

Le premier constat concernant l'outillage GMF est qu'il est nécessaire de répéter les opérations de génération d'éditeur pour chacun d'entre eux. De plus, EuGENia n'ayant qu'une couverture partielle des fonctionnalités offertes par GMF, il a fallu adapter certaines configurations dans les fichiers GMF permettant la génération. Notamment, certaines fonctionnalités reposent sur des déclarations dans le fichier de génération du diagramme d'édition. Comme nous l'avons vu précédemment, ce fichier résulte de la fusion opérée par GMF de ses trois fichiers de déclaration. Dans un contexte de construction itérative d'un DSML, une telle modalité de fonctionnement nuit à la productivité ainsi qu'à l'expérience utilisateur. Néanmoins, les fonctionnalités offertes par GMF permettent de produire rapidement des éditeurs dédiés à un DSML avec des fonctionnalités élaborées. Il est, par contre, plus difficile de procéder à l'ajout de nouveaux comportements en modifiant le code Java généré par GMF. Ceci requiert une bonne compréhension de la pile technologique sur laquelle reposent les éditeurs.

Un cas concret est la mise en œuvre de la fonctionnalité de glisser-déposer d'une sortie d'un modèle exécutable sur un autre modèle exécutable. Comme nous l'exposions au chapitre 3, le comportement attendu pour une telle opération réalisée par un utilisateur CMF est :

- la création d'une nouvelle entrée pour le modèle cible du glisser-déposer ;
- la copie des propriétés de la sortie dans celle la nouvelle entrée ;
- la création d'un lien source-puits entre la sortie et l'entrée et, le cas échéant, la mise en place d'un adaptateur intermédiaire.

Ce type de fonctionnalité est un atout pour le modélisateur, elle peut se rapprocher conceptuellement des fonctionnalités de complétion automatique offertes par un environnement de développement classique. Dans ce contexte, l'opération de glisser-déposer peut être interprétée comme la création d'un lien informel entre deux concepts présents dans le DSML. L'éditeur doit alors permettre de fournir une construction formelle, obéissant à la syntaxe concrète du DSML, qui correspond à l'interprétation de ce lien informel. Pour le cas

de figure évoqué, la réalisation a été rendue difficile par les choix d'implémentation de l'équipe projet de GMF. En effet, au niveau de la représentation graphique, les sorties sont définies comme étant attachées à un modèle exécutable. Cette configuration a pour conséquence qu'une sortie ne peut qu'être déplacée sur le pourtour du modèle auquel elle appartient. Cette fonctionnalité a été générée à un haut niveau dans la hiérarchie de classe de l'éditeur. L'identification de son niveau d'implémentation et des moyens de contournement possibles afin d'obtenir la fonctionnalité escomptée a été difficile à achever. C'est sans doute le reproche majeur qui peut être fait à l'environnement EMF/GEF/GMF, la multiplication des couches technologiques rend difficile la compréhension des niveaux d'intervention possibles pour la personnalisation des éditeurs générés. De plus, l'activité importante des forums utilisateurs ne peut pallier le manque de ressources documentaires accessibles pour ces technologies.

Ceci étant, nous avons obtenu une interface graphique satisfaisante dans le cadre de la réalisation d'un prototype. Les différents éditeurs ont pu être réalisés en employant les formalismes graphiques définis au chapitre 3. De plus, des constructions automatiques sont proposées au modélisateur durant la conception du modèle agronomique. Enfin, le modèle conçu est sérialisé sous la forme d'un fichier XMI, ce qui a permis la mise en place de la génération automatique de code à partir de l'outil Acceleo.

2.2- Intégration de la génération de code

La mise en place de la génération de code a été réalisée en suivant les bonnes pratiques[25] définies par les concepteurs d'Acceleo et en utilisant comme référence le code issu de la réingénierie du modèle de croissance de la vigne. Le système de modules utilisé dans Acceleo se prête particulièrement à cette opération. En effet, il suffit de reprendre le code des classes implémentées pour le projet, de conserver la part du code générique inhérente à l'utilisation du cadriciel et de remplacer les parties spécifiques à un modèle donné par des expressions Acceleo. Les instructions Acceleo sont placés entre crochets ouvrant et fermant : [et /]. La génération de code débute par l'exécution du module principal, ce module est défini par une annotation *@main*. Le code complet du module principal pour CMF est présenté en 4.2. Ce fichier débute par la définition du module *generate* et en spécifiant le métamodèle auquel il s'applique. La référence au métamodèle est donnée en utilisant l'URI qui a été définie dans le

[25] http://www.obeonetwork.com/group/acceleo/page/best-practices (enregistrement requis)

fichier emfatic (voir ci-avant). Cette déclaration est suivie d'une série d'imports d'autres modules. Deux types de modules peuvent être distingués :

- des modules utilitaires comme *javaName,* qui ont pour objectif d'uniformiser les traitements et les générations de chaines de caractères. Dans le cas de *javaName*, nous avons conçu ce module afin de générer, à partir d'un élément du modèle, le nom de la classe Java qui lui correspond ;
- des modules permettant la génération effective de fichiers.

```
[module generate('itk.tooling.model.c3m')]
[import org::ik::generator::cmfgen::main::common::javaName /]
[import org::ik::generator::cmfgen::main::filegen::generateComposite/]
[import org::ik::generator::cmfgen::main::filegen::generateDataProviders /]
[import org::ik::generator::cmfgen::main::filegen::generateInterface4Provider/]
[import org::ik::generator::cmfgen::main::filegen::generateExternalProviderInterface /]
[import org::ik::generator::cmfgen::main::filegen::generateSimulator /]
[import org::ik::generator::cmfgen::main::filegen::generateBlackboard /]
[import org::ik::generator::cmfgen::main::filegen::generateLinker /]
[import org::ik::generator::cmfgen::main::filegen::generateEnumerationForOutputs /]
[import org::ik::generator::cmfgen::main::filegen::generateWorkflow /]
[import org::ik::generator::cmfgen::main::filegen::generateObserver /]
[import org::ik::generator::cmfgen::main::filegen::generateResultHandler /]
[template public generateElement(aSimulator : Simulator)]

[comment @main/]
[generateSimulator(aSimulator)/]
[generateComposite(aSimulator.mainModel)/]
[generateInterface4Provider(aSimulator)/]
[generateExternalProviderInterface(aSimulator)/]

[for (aDP : DataProvider | aSimulator.dataProviders)]
        [generateDataProviders(aDP)/]
[/for]

[generateBlackboard(aSimulator.blackboard)/]
[generateLinker(aSimulator)/]
[generateEnumerationForOutputs(aSimulator)/]
[generateWorkflow(aSimulator)/]
[generateObserver(aSimulator)/]
[generateResHandler(aSimulator)/]
[/template]
```

Code 4.2 : Le module principal de notre projet de génération de code en reprenant la coloration syntaxique de l'éditeur Acceleo

Après ces déclarations, débute la définition du 'template'. Celui-ci est destiné à être exécuté à l'appel du générateur Acceleo. Le template définit une fonction, *generateElement*, qui doit être appliquée à un objet de type *Simulator* issu du modèle transmis au générateur Acceleo. Dans ce template, seules sont présentes des expressions Acceleo. En effet, nous l'avons conçu

pour définir la série de templates à exécuter, ce sont ceux-là qui effectuent la génération de fichiers Java.

Le code 4.3 correspond à la déclaration du template pour la génération de la classe du simulateur. La fonction définie par le template correspond à celle appelée dans le template principal (cf. code 4.2). Associées à la définition de la fonction, les accolades ouvrantes et fermantes sont utilisées pour déclarer des variables locales à l'exécution du template. Cette déclaration permet d'éviter l'emploi d'expressions trop longues dans le template et facilite ainsi la lecture de ce dernier. Après cette déclaration, l'expression entre crochets *file()* indique le début d'une opération de génération d'un fichier. Les paramètres qui lui sont associés sont :

- le chemin complet du fichier ;
- un booléen spécifiant la modalité d'écriture dans le fichier, vrai si le texte est ajouté à celui existant, faux s'il s'agit de le remplacer ;
- l'encodage du fichier.

Notons que le processus de génération est indépendant de la plateforme technologique ciblée. Dans notre cas, nous générons des fichiers dont l'extension est .java mais il serait tout à fait possible de remplacer celle-ci par une extention C++ (.cpp) voire Matlab (.m).

```
[template public generateSimulator(aSimulator : Simulator)
{ packageName : String=fullPackage(aSimulator);
  blackBoardClass : String=aSimulator.blackboard.javaName();
  mainModelClass : String=aSimulator.mainModel.javaName();}]
[file (fullFilePath(aSimulator) +aSimulator.javaName()+'.java', false, 'UTF-8')]
```

Code 4.3 : déclaration du template et de la génération de fichier pour le simulateur

La fonction *fullFilePath* est une fonction que nous avons définie à l'instar de la fonction *javaName*. Dans cet exemple nous pouvons noter deux modalités d'appel différentes pour ces deux fonctions, ces appels sont équivalents. Nous avons utilisé *fullFilePath(aSimulator)* mais aurions pu écrire également *aSimulator.fullFilePath()*. Une troisième notation est possible en appelant simplement *fullFilePath()*, le générateur d'Acceleo cherchera à l'appliquer à l'objet actif. Cependant, cette dernière notation peut-être ambigüe dans le cadre d'une boucle *for* sur des objets, l'objet actif correspondant alors à l'instance obtenue pour chaque itération. Un tel emploi est représenté dans le code 4.4 qui correspond à l'import des différentes classes de *dataproviders* dans la classe simulateur. Dans ce contexte, la fonction *fullPackage* est appliquée à chaque instance de *DataProvider* manipulée dans la boucle.

```
[for (dp : DataProvider | aSimulator.dataProviders) before ('/*Data providers*/\n')]
import [fullPackage()/].[dp.javaName()/];
[/for]
```

Code 4.4 : Génération des imports des *dataproviders* pour la classe du simulateur

Notons que l'application d'une même fonction à différents types de métaclasses est rendue possible par un mécanisme de surcharge des templates géré par Acceleo. Le code 4.5 présente une série de surcharges pour le template *fullFilePath*. Le premier d'entre eux définit un comportement par défaut, le paramètre de type *OclAny* indique que ce template s'applique à n'importe quel type pour lequel une surcharge spécifique n'a pas été définie. Il propose un traitement récursif pour construire le chemin complet du fichier : le template est appliqué à l'élément parent de l'élément passé en paramètre par l'utilisation de la référence *eContainer*. La hiérarchie des éléments du modèle est ainsi remontée jusqu'à atteindre l'élément racine : le simulateur. Pour ce dernier, le chemin d'accès est décrit complètement en prenant en compte le nom du projet en cours.

```
[template public fullFilePath(arg:OclAny)]
[(fullFilePath(arg.eContainer())+arg.javaName().toLower()+'/')/]
[/template]
[template public fullFilePath(alterSM:AlternateSequenceModel)]
[fullFilePath(alterSM.eContainer())/]
[/template]
[template public fullFilePath(sim:Simulator)]
        com/itk/[projectName().toLower().substitute(' ','')/]/simulator/
[/template]
[template public fullFilePath(dp:DataProvider)]
        com/itk/[projectName().toLower().substitute(' ','')/]/simulator/dataproviders/
[/template]
[template public fullFilePath(anInput:Input)]
        [anInput.eContainer(Model).fullFilePath()+'data/input/'/]
[/template]
```

Code 4.5 : La surcharge de template pour une fonction utilitaire définissant un chemin d'accès à un fichier

Outre les templates, l'utilisation de requêtes OCL permise par Acceleo s'est avérée particulièrement utile pour la récupération des différents pas de temps présents dans le modèle agronomique, ainsi que pour obtenir les différents modèles à intégrer dans une séquence d'exécution associée à un pas de temps donné. Nous présentons ci-après (Code 4.6) quelques requêtes OCL que nous avons proposées. Celles-ci permettent un parcours de l'ensemble de la hiérarchie de modèles et la récupération de l'ensemble des modèles exécutables suivant l'ordonnancement établi par la séquence d'exécution. Notons que, parmi les trois requêtes proposées, une seule (*getAllModelsOrderedBySequence*) est de portée publique, les deux

autres sont des requêtes de portée privée permettant de fractionner le traitement. Ces traitements facilitent la génération du code de la classe *Workflow*. Comme nous l'avons vu au chapitre précédent, cette classe gère la séquence d'exécution des modèles en fonction du pas de temps de la simulation.

```
[comment all models ordered by sequence. Sequence is interrupted when an alternate sequence model is met /]
[query public getAllModelsOrderedBySequence(aSimulator : Simulator) : Sequence(Model) =
appendModel(mainModel.firstInFlow) /]


[query private appendModel(mod : Model) : Sequence(Model) =
    if (mod.oclIsKindOf(CompositeModel))
       then appendCompositeModel(mod.oclAsType(CompositeModel))->flatten()
     else if (mod.oclIsKindOf(ExecutableModel))
       then appendExecModel(mod.oclAsType(ExecutableModel))->flatten()
     else Sequence{ self } endif endif /]

[query private appendCompositeModel(mod : CompositeModel) : Sequence(Model) =
    if (nextInFlow.oclIsUndefined())
        then Sequence {appendModel(mod.firstInFlow)->flatten()}
    else (Sequence {appendModel(mod.firstInFlow)}->append(appendModel(mod.nextInFlow)->flatten()))
    endif  /]
```

Code 4.6 : Requêtes OCL pour la récupération ordonnée des modèles exécutables

Lors de la génération de cette classe, il est nécessaire d'effectuer plusieurs appels à cette requête OCL, la gestion du cache des requêtes par Acceleo présente alors tout son intérêt. En effet, celle-ci ne sera exécutée qu'une seule fois lors du processus de génération de code.

Il faut noter que la mise en place de la génération de code a grandement bénéficié de la nature hiérarchique du modèle agronomique. En effet, nous avons pu parcourir l'arborescence qui en résulte et générer les classes en conséquence sans avoir à prendre en compte de possibles références circulaires. Dans le cadre d'autres outils dirigés par les modèles, cette opération pourrait être plus difficile à réaliser.

Nous avons également cherché à intégrer XText à notre fabrique de modèles. Xtext est une solution de l'Eclipse Modeling Project pour la mise en place de DSLs textuels et de leurs éditeurs associés. Xtext est la solution envisagée pour permettre la formalisation de la logique mathématique des modèles exécutables ainsi que pour la saisie des expressions logiques associées à la définition d'une séquence conditionnelle. Il n'existe pas de solution *ad hoc* permettant d'intégrer éditeurs graphiques et textuels pour la mise en place d'un DSML hybride. Le temps imparti au travail de thèse ne nous a pas permis de définir une solution programmatique satisfaisante pour accomplir l'intégration d'éditeurs textuels. Cependant, la solution obtenue rend possible la modélisation conceptuelle formelle d'un modèle grande culture. Nous avons pu en tester les capacités en formalisant les modèles de croissance de la

vigne et du blé. La partie suivante présente le cas d'utilisation de notre prototype pour le modèle de la vigne.

3- Mise en œuvre de CMF avec le modèle vigne

L'éditeur du simulateur (Figure 4.3.1) est le premier éditeur ouvert à la création d'un projet. Ce diagramme contient dès le départ un modèle composite (MainModel) ainsi qu'une référence au *blackboard* (*VinePlot*), leur éditeur respectif s'ouvre par un double-clic. A ce niveau, le modélisateur peut uniquement ajouter ou supprimer des fournisseurs de données externes. Dans le cas présent, une dizaine de fournisseurs sont nécessaires pour le modèle vigne. Notons que la création de fournisseurs de données externes est également possible lors de la déclaration d'une nouvelle entrée pour un modèle exécutable. Une option offre alors la possibilité au modélisateur de créer le fournisseur de données associé à cette nouvelle entrée.

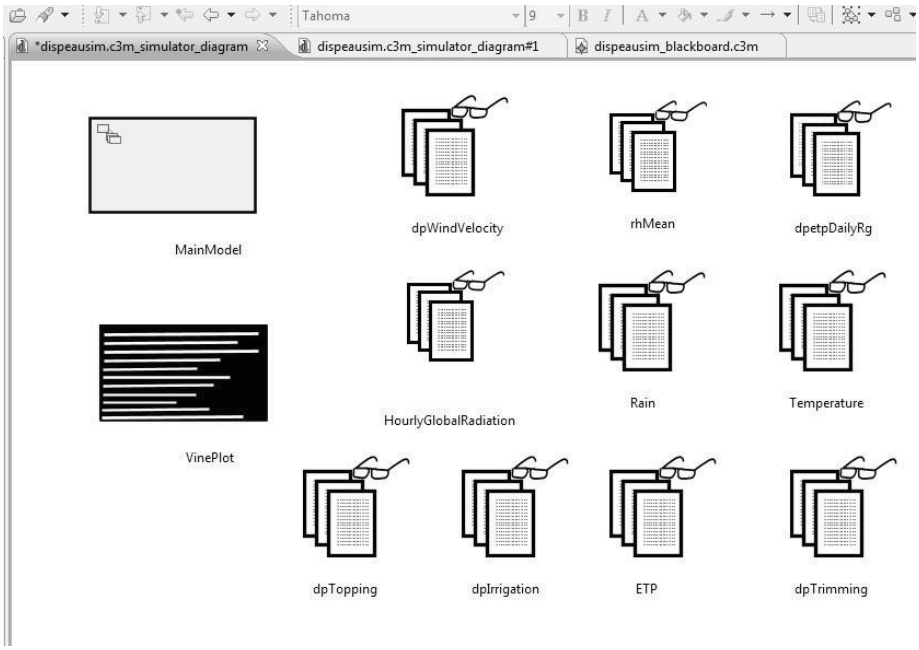


Figure 4.3.1 : Edition du simulateur

L'éditeur du *blackboard* (Figure 4.3.2) a été obtenu directement à partir d'EMF. Il permet par un menu contextuel d'ajouter de nouveaux éléments structurels et de nouvelles variables d'état. Leurs descripteurs sont édités à partir de la fenêtre de propriétés. Ici, la parcelle (*VinePlot*) comporte deux éléments de structure, l'un pour le sol (*soilSystem*) et l'autre pour le pied de vigne (*plantSystem*). Le système sol est décrit à l'aide d'une seule variable d'état indiquant la quantité d'eau disponible dans le sol (*FTSW*). Le système plante est caractérisé

par quatre variables d'état : hauteur (*H*), largeur (*L*), porosité (*Po*) et le *leaf area index* (*LAI*), conformément à la description donnée au chapitre 1.

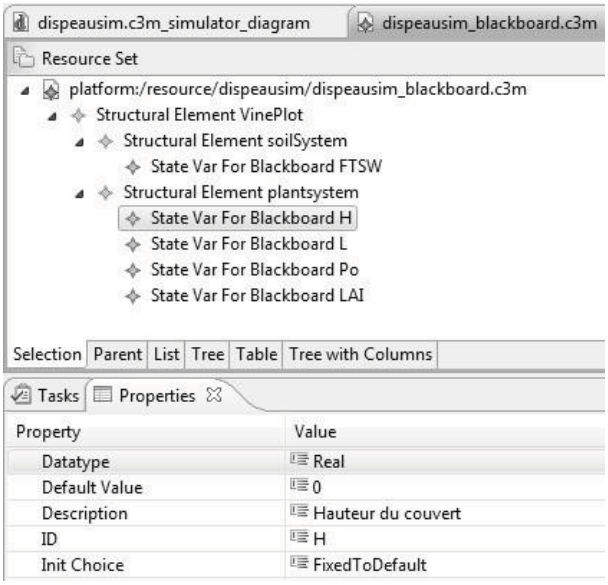


Figure 4.3.2 : Edition du *blackboard*

Les processus en interaction avec le *blackboard* sont définis dans le modèle principal suivant différents niveaux hiérarchiques. La figure 4.3.3 correspond à l'éditeur du modèle principal, cet éditeur est commun à tous les modèles composites. La partie de droite fait apparaître la palette d'outils répartis par modèles et séquences (en haut) et par sources et puits de données (en bas). La syntaxe concrète définie au chapitre 3 été respectée, certains aspects graphiques et fonctionnalités d'affichage restent encore à travailler et à gérer pour un rendu professionnel. L'adaptateur temporel présent dans le diagramme ne fait pas apparaître de flux d'informations sortant. Ceci est dû au fait que le puits de données qu'il alimente est situé à un autre niveau hiérarchique dans le modèle composite VatBrissonFeddes (cf. figure 4.3.4). Ce diagramme met en œuvre le cas de figure évoquée au chapitre 3 (section 2.2) comme exemple justifiant la définition d'un adaptateur conditionnel. En effet, suivant les stations météorologiques, l'ETP (EvapoTranspiration Potentielle) sera fournie ou non comme une donnée environnementale à l'aide d'un fournisseur de données externes. En son absence, le calcul de l'ETP est nécessaire pour déterminer la perte d'eau réelle par le système plante-sol. La séquence débute donc par une séquence conditionnelle déterminant la voie à suivre en fonction de la fourniture de l'ETP. L'adaptateur conditionnel est en charge de fournir la bonne source de données en entrée du modèle calculant l'évapotranspiration réelle.

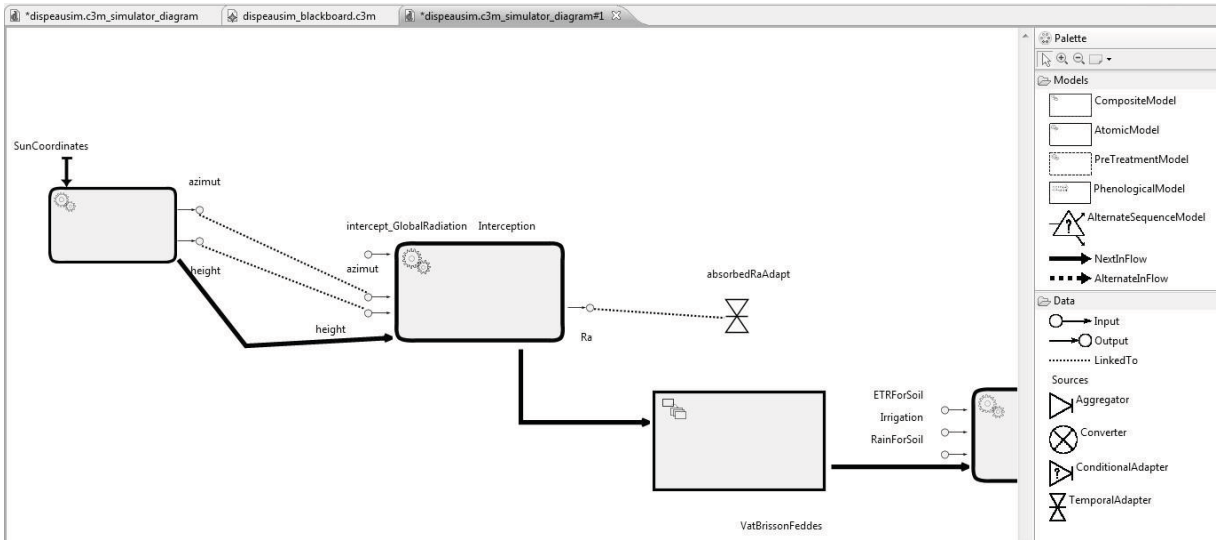


Figure 4.3.3 : Edition du modèle principal

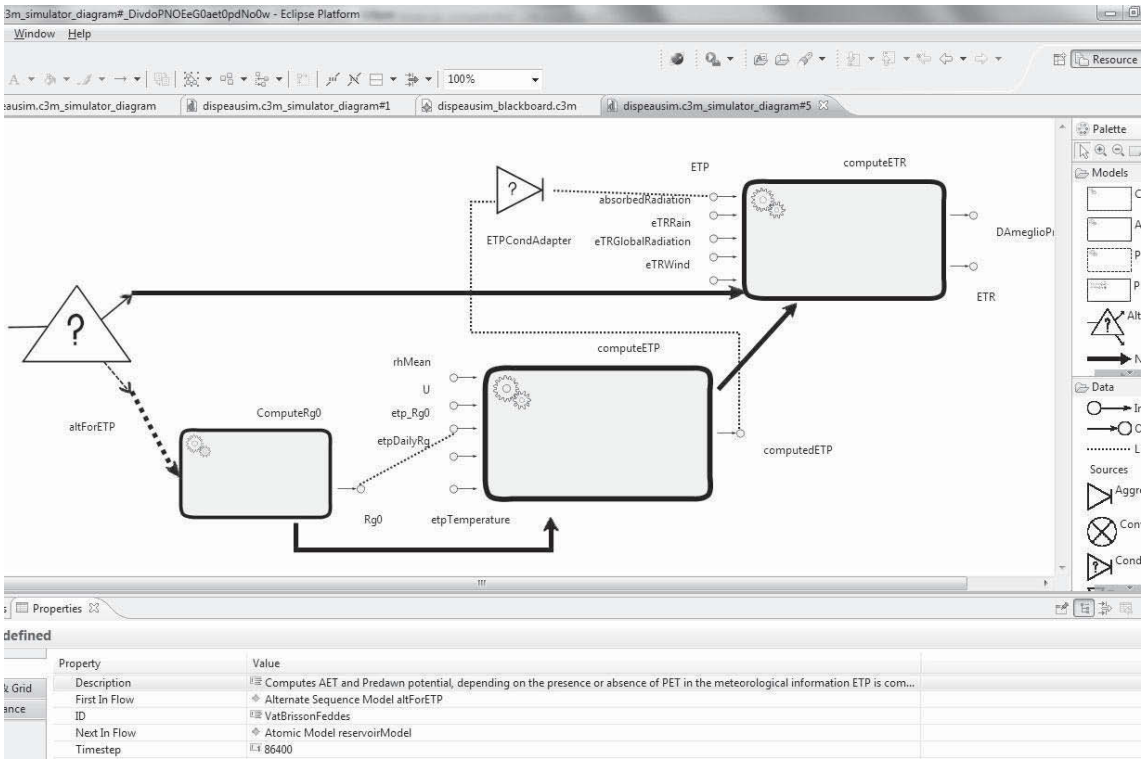


Figure 4.3.4 : Edition du modèle transfert d'eau plante atmosphère (Barbier *et al.* 2013a - In press)

La représentation de la croissance et du développement (Figure 4.3.5) est assez simple, elle n'est gouvernée que par l'effet des températures et d'éventuelles interventions de taille du couvert végétal (*topping* et *trimming*). Le diagramme du modèle phénologique (Figure 4.3.6) présente les huit stades consécutivement rencontrés par la vigne durant la saison. Ce diagramme devra être retravaillé pour une meilleure présentation graphique mais surtout pour pouvoir intégrer les différentes fonctions de la température moyenne permettant de déterminer l'avancement du stade phénologique et la transition vers le stade suivant. Cette fonctionnalité est connexe avec les notions de réutilisabilité et d'exécutabilité des modèles comme nous le verrons en perspectives.

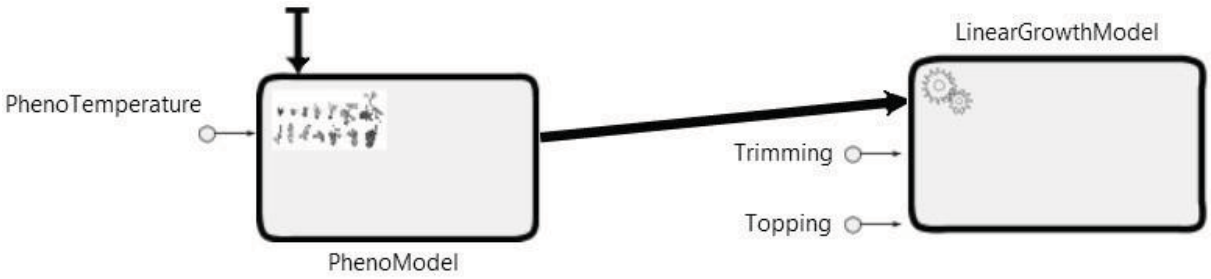


Figure 4.3.5 : Croissance et développement de la plante dans le modèle vigne

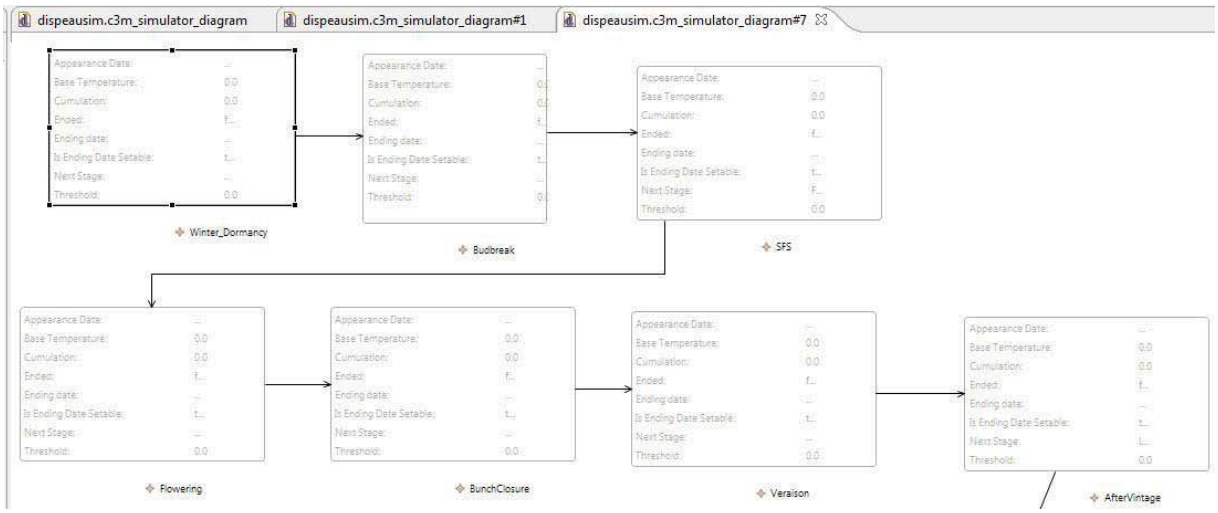


Figure 4.3.6 : Le modèle phénologique pour la vigne

La formalisation graphique du modèle vigne a été accomplie à l'aide de soixante concepts graphiques. Ces éléments sont répartis dans six diagrammes, deux d'entre eux ne sont pas représentés dans les figures : le modèle phénologique et le modèle de croissance. Ces diagrammes présentent l'avantage d'offrir une représentation synthétique du modèle et sont un atout pour la (re)découverte du modèle. A partir de cette conception visuelle du modèle, nous avons procédé à la génération du code correspondant. Cette génération a permis de

produire 1500 lignes de codes auxquelles s'ajoute la documentation du code par la reprise des différentes descriptions des modèles et de leurs constituants. Une partie du programme reste à développer manuellement. Nous avons comme perspective le développement de la partie textuelle de notre DSML qui permettra d'automatiser encore plus la génération de code avec notamment :

- l'implémentation effective des méthodes de simulation de chaque modèle exécutable ;
- la définition des expressions logiques associées aux adaptateurs et séquences conditionnels.

Pour le modèle vigne, cette étape manuelle complémentaire a conduit à l'écriture de 300 lignes de code (soit 16% du total). Elle a été grandement facilitée par la correspondance hiérarchique entre le code généré et les modèles du prototype Matlab. On peut retenir que près de 85% du Java code est généré automatiquement grâce à l'outillage logiciel proposé.

4- Discussion et perspectives

Notre travail a été mené afin de répondre à une problématique industrielle précise. Dans le cadre de la production de modèles grande culture pour la réalisation d'outils d'aide à la décision, les équipes d'ITK mettent en place des prototypes de modèles agronomiques dans un environnement Matlab. Ces prototypes sont utilisés comme référence pour la mise en place d'implémentations Java de ces modèles. L'absence, par manque d'outillage adéquat, de documentation formelle des modèles et l'existence de leur double implémentation sont problématiques en termes de coûts de production. Elles présentent également un risque de disjonction entre les deux implémentations dans le cadre de l'évolution du modèle agronomique.

Pour permettre d'améliorer le processus de production de nouveaux modèles agronomiques, nous avons proposé la mise en œuvre des technologies de l'Ingénierie Dirigée par les Modèles (IDM) afin de concevoir une fabrique de modèles agronomiques : CMF. Celle-ci est destinée à la formalisation des modèles conceptuels de manière graphique. La formalisation d'expressions mathématiques et logiques par un langage spécifique dont la syntaxe concrète est textuelle doit amener à l'obtention d'un environnement complet de Modélisation et Simulation (M&S) doté de capacités de génération de code Java. Dans cette optique, nos travaux se sont focalisés sur le prototypage de CMF en y intégrant la partie graphique du DSML (Domain-Specific Modeling Language) envisagé. Le prototype obtenu est doté d'une

fonction de génération de code qui, en s'appuyant sur le cadriciel que nous avons défini, permet d'obtenir le code Java correspondant au modèle conceptuel formalisé dans CMF.

Certains avantages intéressants se dégagent de notre réalisation. En premier lieu, bien qu'il ne s'agisse que d'un prototype, le modèle conceptuel offre une facilité de lecture et de réappropriation d'un modèle existant. En fin de thèse, des agronomes ont repris le modèle de blé afin d'y apporter des améliorations, pour procéder au passage de connaissances, nous nous sommes appuyés sur le modèle conceptuel défini à l'aide de CMF. Cela nous a permis de retrouver rapidement les caractéristiques principales de ce modèle ainsi que ses particularités. Ainsi, nous retrouvons les propriétés originelles du modèle conceptuel qui permet la documentation d'un modèle scientifique.

En second lieu, la génération automatique du code Java à partir du modèle conceptuel correspond à la majeure partie du code nécessaire à l'obtention de l'implémentation d'un modèle agronomique fonctionnel. Ayant pratiqué la traduction de code Matlab vers le code Java ainsi qu'étudié les implémentations mises en place par d'autres développeurs, nous avons constaté que la définition de la séquence d'exécution et la transmission des informations présentent le plus grand risque d'erreur lors de la traduction. Certes, l'élément central qu'est la formalisation de la logique des processus biophysiques fait encore défaut. Cependant, il s'agit d'une partie de l'implémentation dont la longueur est modérée. De plus, nos choix de conception conduisent à une correspondance parfaite entre la représentation d'un processus et une fonction Matlab. Cette correspondance n'est pas due au hasard : à l'issue de la première année de thèse nous avons participé à l'orientation des normes de développement vers l'adoption d'une structure modulaire pour les prototypes réalisés par les modélisateurs agronomes.

Enfin, de récents travaux ont été menés sur l'implémentation Java du modèle vigne issue de la réingénierie que nous avons menée. Ces travaux ont été conduits par un binôme ingénieur logiciel et modélisatrice. Leur objectif était de remplacer le module de croissance et d'adapter certaines formalisations de processus préexistants. Plusieurs constats ont été réalisés à l'issue de ces travaux :

- pour l'ingénieur logiciel, l'ajout d'un nouveau formalisme au modèle Java se réalise aisément sans qu'il soit nécessaire de comprendre l'ensemble des tenants et aboutissants du cadriciel que nous avons conçu ;

- pour la modélisatrice, la structuration du code permet une compréhension de l'implémentation sans nécessaire formation aux concepts de la programmation orientée ;
- ces deux points ont abouti à une évolution du modèle avec une validation de ses sorties par rapport au prototype Matlab en à peine une heure trente minutes. La facilité et la rapidité de la réalisation ont été particulièrement appréciées par le binôme.

En somme, cette première expérience offre des signaux encourageants quant à nos choix de conception. Il paraît nécessaire de confirmer sur la durée ces premiers retours par la réitération de ce type d'activité.

Les modélisateurs agronomes ont été impliqués, lors de plusieurs réunions, sur la définition de maquettes pour l'interface de CMF. Cependant, l'un des reproches qui peuvent être fait concernant notre démarche est que CMF n'ait pas encore été directement employée par les modélisateurs agronomes. Cela résulte de notre volonté : en effet, les recommandations de Steven Kelly et Risto Pohjonen (Kelly et Pohjonen 2009) sont de ne pas négliger la formation des utilisateurs sous peine de compromettre sur la durée l'adoption du DSML. Or, après l'obtention du prototype, notre planning de thèse ne nous a pas permis de consacrer le temps nécessaire à la formation des utilisateurs. Nous avons donc préféré reporter la mise à disposition du prototype afin de pouvoir assister les modélisateurs d'ITK dans sa prise en main. Il est d'ores et déjà prévu d'utiliser CMF pour les prochains modèles mis en place par ITK afin de procéder à l'évaluation en situation réelle de notre prototype.

Il est vraisemblable que cette phase de test conduise à des demandes d'évolution de CMF. Certaines sont déjà prévisibles, elles tiennent à des contraintes technologiques issues de GMF (Graphical Modeling Framework), de choix par défaut que nous avons pu réaliser ou d'éléments que nous n'avons pas pu intégrer au cours du travail de thèse.

4-1- Evolution technologique

Certains comportements issus de l'utilisation de plusieurs diagrammes produits par GMF ne nous paraissent pas adaptés. En effet, dans certains cas la modification et l'enregistrement d'un diagramme d'un modèle composite peut conduire à ce que le diagramme de niveau hiérarchique supérieur passe dans un état modifié nécessitant à nouveau la sauvegarde par l'utilisateur. Ce comportement est peu naturel et risque de susciter l'incompréhension des

utilisateurs. Il paraît nécessaire de trouver une solution technique afin de l'adapter aux attentes des modélisateurs agronomes.

D'un point de vue expérience utilisateur, d'autres éléments associés à l'utilisation de GMF pour obtenir l'interface graphique posent question. Comme le montrent nos exemples d'utilisation de CMF pour le modèle vigne, nous avons pu intégrer des figures au format SVG (Scalable Vector Graphics) pour définir la syntaxe concrète de notre DSML. Dans les éditeurs obtenus à l'aide de GMF, le redimensionnement des formalismes visuels est pris en charge par défaut. Cependant, si l'agrandissement des images se déroule correctement, leur rétrécissement pose problème. Dans la majeure partie des cas, il est impossible de le réaliser, l'image gardant alors sa taille initiale. Si ce n'est qu'un point de détail, cela montre un manque de maturité de la solution GMF qui peut amener les futurs utilisateurs du DSML à remettre en cause sa qualité générale. Toujours sur les aspects de représentation graphique, la palette d'outils, permettant d'ajouter de nouveaux concepts dans le modèle, est configurée à l'aide de GMF afin de présenter les concepts utilisables sous la forme d'icônes. Cependant, les images acceptées à la configuration ont une résolution maximale de 48*48 pixels. Il en résulte un rendu graphique qui n'est pas acceptable dans un contexte professionnel. De même, le repositionnement d'éléments du diagramme est trop limité, plus particulièrement le déplacement des étiquettes, telles que celles représentant des entrées ou sorties, manque de précision. A partir de quatre entrées pour un modèle, il devient difficile de visualiser la bonne association étiquette/formalisme visuelle. Pour résumer, GMF est un outil puissant mais ses capacités d'intégration d'éléments visuels sont assez limitées. Elles laissent penser que ces fonctionnalités ne font pas partie des priorités des concepteurs. Pourtant l'apparence de l'interface est un critère important pour son acceptation par les utilisateurs ciblés. De plus, doter un langage visuel d'éléments visuels aisément compréhensibles est essentiel. Bran Selic (Selic 2003) souligne l'importance du choix de la notation pour faciliter cette compréhension, ce choix ne devrait pas être contraint par l'environnement utilisé pour concevoir le langage.

D'un point de vue technique, l'adaptation des éditeurs obtenus à partir de GMF nous semble avoir un coût trop élevé en termes de coût de production. De plus, le processus de production des différents diagrammes est assez surprenant. Le fait qu'il n'existe pas de solution *ad hoc* pour produire tout un jeu d'éditeurs à partir d'un unique métamodèle nous paraît contre-productif. Ce système oblige à la répétition d'opérations de génération avec une multiplication des générations des classes spécifiques au métamodèle alors que le résultat est à l'identique. De plus, la liaison des éditeurs entre eux doit être effectuée à la main dans le code par la mise

en commun du contexte transactionnel. Si cette dernière opération n'est pas très complexe à réaliser, elle pourrait être accomplie automatiquement lors de la génération de l'ensemble des éditeurs nécessaires pour un DSML donné. Ne disposant pas de l'explication justifiant les choix des concepteurs du Graphical Modelling Project, nous ne pouvons qu'avancer des interprétations.

Tout d'abord, GMF pâtit peut-être de son historique de développement. Sa première sortie date de 2006 dans la version Ganymède d'Eclipse. Il est conçu pour exploiter EMF et pour faciliter l'utilisation du Graphical Editing Framework (GEF). Ces deux projets évoluant au cours du temps, des adaptations de GMF ont peut-être été nécessaires ce qui peut engendrer une dette technique sur le projet. Pour ce dernier point, il nous paraît important de souligner que, malgré la volonté de la fondation Eclipse de créer un projet dédié aux technologies dirigées par les modèles : EMP (Eclipse Modelling Project), ce projet est avant tout le rassemblement sous une même bannière de différents outils sans qu'une démarche fédérative se dégage réellement. En effet, particulièrement sur les aspects d'interfaçage de différents outils appartenant à l'EMP, les ressources documentaires sont assez limitées et bien souvent obsolètes. L'organisation des forums utilisateurs, seule source d'informations réellement à jour, est faite par outil ou sous-projet. Cela ne permet pas aisément l'identification des solutions à appliquer pour les activités d'interfaçage.

Ensuite, il existe peut-être une vision quelque peu réductrice de la modélisation à l'aide d'un langage à syntaxe concrète graphique. Les formalismes accessibles par défaut ne sont pas sans faire penser aux formalismes UML. Ainsi, il existerait une influence des pratiques de conception orientée objet dans la conception de l'outillage permettant l'obtention de solutions de modélisation graphique spécifiques à un domaine. Bien entendu, cela n'empêche pas la mise en place de solutions évoluées telles que Talend Open Studio®[26], outil de migration de bases de données réalisé à partir d'EMF/GEF/GMF. Cependant, le coût de réalisation de tels outils reste élevé. Ce travers pourrait être évité par une vision plus large des possibilités de création de langage graphique et favoriser, de fait, l'émergence de nouveaux outils de conception basés sur des DSMLs. Dans (Barbier *et al.* 2013b - In press), nous suggérons de procéder à la métamodélisation du patron de conception modèle-vue-contrôleur afin d'obtenir un DSML permettant la définition de nouveaux DSMLs. Nous n'avons pas pu explorer cette piste, elle pourrait s'avérer intéressante pour faciliter la création de nouveaux langages et,

[26] http://www.talend.com/products/talend-open-studio

surtout, elle montrerait que la communauté IDM applique ses propres paradigmes aux environnements qu'elle conçoit. Une telle démarche aurait donc un impact promotionnel pour une adoption plus large de l'ingénierie dirigée par les modèles.

Dans ces conditions, il est possible que nous passions à un autre environnement technologique pour l'obtention d'une fabrique de modèles grande culture plus satisfaisante d'un point de vue graphique et ergonomique. En effet, de ce point de vue, nos objectifs de réalisation sont relativement simples et le nombre de concepts présents dans notre métamodèle est assez limité. Plusieurs pistes sont envisageables, la première pourrait être de réaliser notre propre solution gérant la conception de modèles et leur persistance. La liaison avec le métamodèle C3M ne s'opérant alors que sur les aspects de sérialisation de modèles sous la forme d'un fichier XMI. Cela permettrait de conserver la partie génération code à l'aide d'Acceleo dont nous sommes satisfaits. Cependant, une telle solution nous empêcherait vraisemblablement d'utiliser les fonctionnalités de débogage et de vérification dont nous pouvons bénéficier avec l'emploi d'EMF.

Une possibilité intermédiaire pourrait donc reposer sur l'emploi des classes du métamodèle obtenues à partir d'EMF en lien avec une interface développée par nos soins. Une troisième piste présente une bonne alternative à l'emploi de GMF : Graphiti[27]. Cet outil fait partie du Graphical Modeling Project au même titre que GMF. Il n'est encore pour l'instant qu'en phase d'incubation mais a beaucoup évolué et se rapproche de sa première version. Graphiti est conçu pour la définition de DSMLs à syntaxe concrète graphique et exploite également EMF et GEF. A la différence de GMF, ses contributeurs ont pour objectif de masquer la complexité technologique de GEF. Ce choix peut avoir un impact sur les possibilités de personnalisation de l'interface obtenue. Il nous faudra donc évaluer si les éditeurs obtenus à partir de Graphiti répondent à nos attentes. En tout état de cause, il nous paraît indispensable de conserver un environnement basé sur le langage de programmation Java. En effet, grâce à cela nous pouvons envisager d'intégrer des fonctionnalités permettant au modélisateur agronome d'enrichir son environnement de conception comme nous le verrons dans la partie suivante.

[27] http://www.eclipse.org/graphiti/

4.2- Intégration d'une syntaxe concrète textuelle

La conception de modèles grande culture à l'aide de CMF ne saurait être complète sans l'intégration à CMF d'éditeurs de texte spécifiques. Ceux-ci doivent permettre aux modélisateurs de définir les expressions logiques et mathématiques correspondant à la représentation d'un processus biophysique donné. Ces expressions déterminent le comportement d'un modèle exécutable. Par défaut, EMF ne permet pas la spécification du comportement, tout au plus accepte-t-il la définition de l'interface correspondant à l'opération (Steinberg *et al.* 2008). Après génération, il est possible de procéder à l'implémentation en Java de cette dernière.

Dans le cadre de notre métamodèle, l'implémentation Java de la classe modèle exécutable contient l'ensemble des éléments qui caractérisent ce type de modèles : ses entrées, sorties, variables d'état et paramètres sous la forme de collections (*EList*). Cette classe dispose également d'une méthode *execute()* qu'il est possible d'implémenter. Une telle conception suppose donc que tout modèle exécutable ait un comportement à l'identique de tous les autres. Ceci rejoint la vision exposée par Estublier et ses coauteurs (Estublier *et al.* 2005) page 74 : « An important characteristic of most DSLs is that the metamodel encapsulates most (if not all) the behavior of the entities in the domain. (…) Most of the time the model represents only the structural part of the application and simply parameterizes the predefined behavior. The fact a model is purely structural has important consequences. »[28]. De la plupart des cas où le modèle est structurel, les auteurs en arrivent au fait qu'un modèle soit purement structurel. La même philosophie est employée dans l'Eclipse Modeling Project puisque ce choix est défini dans EMF qui est la base de l'outillage du projet de modélisation d'Eclipse. La logique est donc de couvrir la majeure partie des besoins en termes de modélisation et d'adapter le code obtenu par génération. Si l'on peut comprendre ce raisonnement, il n'en reste pas moins qu'il exclue tous les cas de figures où le spécialiste du domaine est le dépositaire de tout ou partie du comportement des entités et qu'il lui revient donc de spécifier. Tel est le cas pour notre fabrique de modèles agronomiques.

[28] Traduction libre de notre part : « Une caractéristique importante de la plupart des DSLs est que le métamodèle contient la plupart (si ce n'est tout) des comportements associés aux entités du domaine. (…) La plupart du temps le modèle ne représente que la partie structurelle de l'application et se contente de paramétrer le comportement prédéfini. Le fait qu'un modèle soit purement structurel a des conséquences importantes. »

Dans le contexte de CMF, une partie du comportement des entités est prédéfinie, ce qui justifie la mise en place d'un cadriciel support de la génération automatique de code. Cependant, ce cadriciel permet avant tout l'intégration numérique de résultats de fonctions dont la formalisation dépend du modélisateur. Ainsi, tous les membres (attributs et opérations) d'une instance de la métaclasse *ModeleExecutable* peuvent être manipulés. Il faudrait donc fournir au modélisateur agronome une possibilité de modifier l'implémentation de la méthode *execute()* au niveau instance.

La syntaxe envisagée est constituée de quelques mots clés listés dans le tableau 4.7, la signification de chacun des éléments de syntaxe est définie dans la deuxième colonne. Certains mots sont réservés à l'appel d'instances partagées par l'ensemble des modèles, il s'agit de l'horloge (*clock*), du *blackboard* et du stade phénologique courant (*currentStage*). Pour l'horloge, une série de fonctions est accessible pour obtenir différentes représentations du temps ou opérer des comparaisons de dates. A partir du *blackboard*, il est possible de :

- parcourir l'ensemble de ses éléments structurels ;
- accéder en lecture ou en écriture à ses variables d'état ;
- créer de nouveaux éléments structurels par une méthode *createNOM()*, où *NOM* correspond à la dénomination de l'élément structurel à créer (p.ex. : *Leaf*). Comme nous l'expliquions au chapitre 3 (partie 4.2.3), le paramétrage de cette méthode est conditionné par les règles d'initialisation des variables d'état associées à cet élément structurel.

Le stade phénologique courant - c'est-à-dire au temps de la simulation – est également accessible depuis le *blackboard*, il paraît cependant intéressant de pouvoir y accéder directement sans passer par ce dernier afin de faciliter la tâche du modélisateur.

Les boucles classiques de la programmation (*for, while*) ainsi que les instructions conditionnelles sont nécessaires (*if...else*). De plus, nous souhaitons intégrer une boucle *foreach* destinée à parcourir une collection d'éléments structurels issus du *blackboard*. A cela, s'ajoutent les opérateurs mathématiques classiques d'addition, soustraction, multiplication, division, puissance et exponentielle. Enfin un ensemble de fonctions mathématiques (non listées dans le tableau) est disponible telles que sinus, cosinus, maximum et minimum.

Mots-clés	Signification
blackboard	Accès direct au blackboard des accesseurs permettent de parcourir les éléments structurels, les variables d'état sont accessibles en lecture/écriture
clock	Accès à différentes représentations du temps au cours de la simulation et à des fonctions de comparaison de dates
currentStage	Accès au stade phénologique actuel

for (i=a ; i<b) {
…
}
	Boucle itérative avec incrément de 1 sur la valeur i, a et b valeurs entières et a inférieur ou égal b. A priori la spécification d'une autre valeur pour l'incrément n'est pas nécessaire. Autre notation possible *for (i= a, b)*

if (….)
else
endif
	Opérateur conditionnel si / sinon

while (…) { }	Boucle while répétition tant que la condition exprimée est vraie
foreach (ChildElement _elem in Element)	Boucle permettant de parcourir une collection de sous-éléments du *blackboard*
int i, real r, bool b, string s	Déclaration de variables locales
=	Opérateur d'affectation
>, <, <=, >=, ==, !=	Opérateurs de comparaison
+, -, *, /, ^, exp	Opérateurs mathématiques : addition, soustraction, multiplication, division, puissance et exponentielle

Tableau 4.7 : Eléments de la syntaxe concrète textuelle envisagée pour CMF

Cette syntaxe permet également la création de variables locales, l'utilisation de types entiers (*int*) et réels (*real*) nous paraît indispensable. Le besoin de déclarer des variables locales booléennes (*bool*) ou de type chaîne de caractères (*String*) n'est pas encore confirmé : nous n'avons pas eu à employer ce type de variables pour les modèles blé et vigne mais elles peuvent s'avérer nécessaires pour d'autres modèles. Il faut par contre inclure la possibilité de créer une référence locale à un élément structurel du *blackboard*.

Concernant les opérateurs, certaines vérifications devront être exercées. Outre la compatibilité de type classique dans un langage de programmation standard :

- les entrées d'un modèle ne pourront être utilisées que dans la partie droite d'une opération d'affectation ;
- les sorties ne sont présentes que dans la partie gauche d'une opération d'affectation.

De plus, les opérateurs de comparaison doivent permettre de comparer le stade phénologique courant avec l'un des stades phénologiques pris en compte par le modèle.

D'un point de vue technique, plusieurs possibilités sont envisageables pour parvenir à l'obtention d'éditeurs permettant de manipuler une telle syntaxe. Ce sont des pistes que nous devons encore évaluer. En premier lieu, il serait possible de donner une implémentation commune à tous les modèles exécutables de la méthode *execute()* et de transférer le comportement spécifique à un attribut de type chaîne de caractères. Son édition serait déléguée à un éditeur Xtext obtenu à partir de la spécification de notre syntaxe textuelle. Quant à la méthode *execute()*, elle ferait appel à un interpréteur obtenu depuis Xtext et adapté au modèle en cours d'exécution. Une autre possibilité serait d'utiliser Kermeta (Muller *et al.* 2005) conçu pour permettre la spécification du comportement. Il présente l'avantage de pouvoir être intégré avec des éditeurs obtenus à partir de GMF et d'être également basé sur le langage Java.

Ce dernier point est important. En effet, nous souhaiterions permettre à terme l'import par les modélisateurs de bibliothèques supplémentaires afin de bénéficier, par exemple, de fonctions de résolution numérique évoluées. L'ajout de la référence à de telles bibliothèques paraît réalisable pour les éditeurs textuels, cependant leur utilisation doit également être transférée dans le code Java généré à l'aide d'Acceleo. En conservant le Java comme langage généraliste sur lequel repose CMF, les bibliothèques importées seront des bibliothèques Java ou compatibles avec ce langage et pourront donc être transférées vers le code généré.

4.3- Simulation du modèle grande culture dans CMF

Une fois la syntaxe concrète textuelle obtenue, il sera possible de procéder à la mise en œuvre de l'exécutabilité, c'est-à-dire permettre la simulation du modèle grande culture dans l'environnement CMF. Les choix techniques permettant d'obtenir la fonctionnalité de simulation dépendront de ceux réaliser pour la mise en place de la syntaxe concrète textuelle. De plus, la modalité d'exécution retenue aura également des conséquences. En effet, il nous serait possible de procéder à l'interprétation du modèle en mémoire ou de générer le code Java du modèle et d'exécuter ce dernier. Nous privilégions plutôt le mécanisme interprétatif. Avec une implémentation correcte, ce dernier devrait être plus rapide que la double opération génération/exécution. En effet, les tests de génération de la structure du modèle vigne et du modèle blé se sont déroulés en cinq secondes en moyenne. Si à la structure se rajoute la partie définissant le comportement des modèles, il est tout à fait envisageable que cette génération

demande une dizaine de secondes. Un tel délai entre le lancement de la simulation et l'obtention du résultat ne saurait satisfaire le modélisateur agronome.

Parmi les impacts prévisibles, le métamodèle devra être modifié pour permettre la réalisation de la simulation. En effet, tel que nous l'avons présenté, les puits et sources de données ne disposent pas d'attribut permettant de stocker l'information produite ou utilisée lors de la simulation. De plus, le typage de cette information doit être géré. Plusieurs choix de conception sont possibles. Le premier serait de définir, pour chaque type de puits et de sources de données, des sous-classes spécifiques au type d'information attendu, p.ex : *IntegerInput*, *RealInput*. Le deuxième, que nous pensons privilégier, est représenté par la figure 4.4.3. L'idée serait d'associer aux sources de données une métaclasse *Data* et de définir des sous-classes typées dotées d'un attribut valeur : *IntegerData*, *RealData*. La relation entre sources et puits de données étant bidirectionnelle, à l'exécution, un puits de données accède à la source de données qui l'alimente et obtient la valeur produite par cette source.

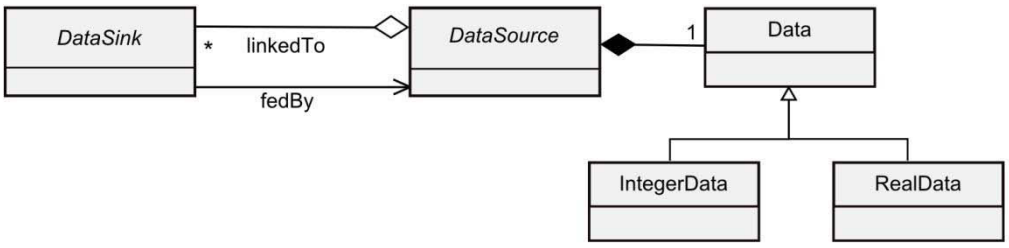


Figure 4.4.3 : Proposition d'évolution du métamodèle pour permettre la transmission d'informations lors de la simulation dans l'environnement CMF

Une autre partie du métamodèle est inadaptée à la simulation, il s'agit des métaclasses conçues pour décrire le *blackboard*. Nous ne souhaitons pas remettre en cause sa représentation au niveau méta. Cependant, la représentation que nous donnions au chapitre 3 reprise dans la figure 4.4.4 ne peut être utilisée directement pour la simulation. En effet, dans ce contexte, le modèle en mémoire contient un élément structurel identifié comme étant une tige. Cet élément structurel possède un unique élément fils phytomère associé à une cardinalité *1..n*. Or l'objectif de cette déclaration n'est pas de définir un unique élément phytomère associé à la tige mais une collection d'éléments de ce type qui puisse être manipulée et enrichie au cours de la simulation. Il serait possible d'enrichir l'implémentation des éléments structurels pour permettre ce type de comportement.

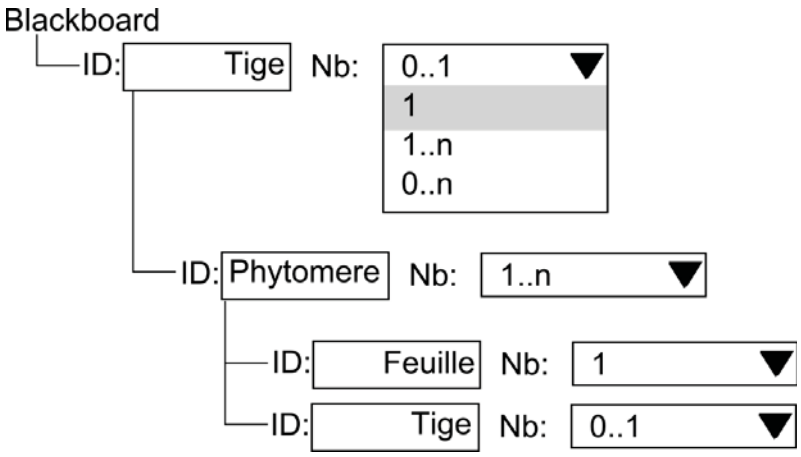


Figure 4.4.4 : Edition du *blackboard*, cas concret pour la mise en évidence des éléments structurels (reprise du chapitre 3)

Pour autant, nous pensons plutôt procéder, pour ce cas précis, à la génération du code correspondant au *blackboard.* Cette génération serait déclenchée systématiquement suite à la sauvegarde demandée depuis l'éditeur du *blackboard*. Son contenu est de taille limitée, la génération du code associée devrait être assez rapide, de plus elle doit pouvoir être réalisée sans empêcher le modélisateur d'effectuer d'autres actions. Enfin, l'avantage majeur qui pourrait résulter de cette fonctionnalité est l'obtention des accesseurs nécessaires à la manipulation des éléments du *blackboard* ainsi que la génération des méthodes permettant de créer de nouveaux éléments telles que spécifiées ci-avant. Il faudrait ensuite mettre à disposition des éditeurs textuels l'implémentation générée.

Enfin, certaines classes techniques seront mises en place. Nous utilisons le terme de classes techniques car elles ne sont pas modifiées directement par le modélisateur. Cependant nous n'excluons pas de les intégrer dans le métamodèle. L'une d'entre elles est destinée à jouer le rôle du *Workflow* tel qu'il est défini dans le cadriciel. Sa conception est fortement liée aux différents choix opérés concernant la syntaxe textuelle. Dans ces conditions, il est difficile de prévoir la forme que prendra son implémentation. Certaines classes seront également nécessaires pour lire les fichiers de données météorologiques afin d'alimenter les fournisseurs de données externes. Enfin, le résultat de la simulation devrait être délégué à une classe similaire au *ResultsHandler* du cadriciel. Ces résultats devront être gardés en mémoire afin d'offrir au modélisateur la possibilité de les explorer ou d'en obtenir des représentations graphiques.

Ces différents éléments mèneront à l'utilisation de CMF comme un environnement de modélisation et de simulation pour les agronomes, ce qui devrait résulter à terme en l'abandon

de l'utilisation de Matlab pour la conception des prototypes de modèles grande culture. Or, comme nous l'expliquions au chapitre 3 (partie 1.2), au cours du prototypage, le modélisateur peut tester différents formalismes pour un même processus biophysique en recourant à des configurations différentes du simulateur. Cette fonctionnalité est nécessaire pour CMF et doit être prise en compte au moment du lancement de la simulation. En effet, le modélisateur devra disposer d'une interface proposant de sélectionner la configuration qu'il souhaite utiliser lors de la simulation. Cela peut nous conduire à intégrer dans le métamodèle une métaclasse fille de l'*AlternateSequenceModel* qui soit spécifiquement dédiée à la gestion de la configuration. De plus, cette fonctionnalité aura un impact sur la génération du code Java destiné à être mis en production. Lorsque différentes configurations sont disponibles, la (les) configuration(s) à intégrer dans le code généré devra (devront) être définie(s).

4.4- Vérification et validation

L'obtention de la syntaxe textuelle ainsi que l'exécutabilité des modèles dans l'environnement CMF sont les conditions indispensables à la réalisation des objectifs, fixés au chapitre 1, que sont l'obtention :

- d'un environnement de modélisation et simulation pour les modélisateurs d'ITK ;
- d'une implémentation Java des modèles grande culture de l'entreprise sans intervention d'ingénieurs en génie logiciel.

Un dernier point reste à soulever concernant les objectifs que nous fixions : la validation de l'implémentation Java générée automatiquement. Dans le domaine de l'Ingénierie Dirigée par les Modèles (IDM), la validation des approches génératives restent un sujet de recherche actif (Kornecki et Johri 2006; Merilinna et Pärssinen 2010). Comme nous l'exposions au chapitre 2 (partie 1.3), il n'existe pas de solution *ad hoc* permettant la validation du code généré. Toujours dans cette partie, nous expliquions que l'absence de solution adaptée à l'ensemble des DSMLs est inhérente à la spécificité de chacun d'entre eux et au domaine auquel il s'applique.

Dans le cas de CMF, la particularité est que l'expert du domaine ne saurait définir à l'avance le comportement attendu. En effet, le modèle scientifique est destiné à être simulé afin de produire, grâce aux capacités de calcul de l'ordinateur, des informations que le cerveau humain ne saurait fournir seul. Le modélisateur construit le prototype, utilise sa connaissance experte du domaine pour évaluer le comportement de celui-ci et confronte les résultats de

simulation à des données obtenues sur le terrain. Après obtention de résultats satisfaisant les objectifs qu'il s'est fixé, le modélisateur peut alors procéder à la génération de l'implémentation Java. Pour valider cette étape, nous proposons d'intégrer à CMF une fonctionnalité de définition de scenarii de test contenant : les fichiers météorologiques, les fichiers de paramètres ainsi que les sorties obtenues lors de la simulation dans l'environnement CMF. Après la génération de code, le simulateur obtenu serait automatiquement confronté à ces scenarii. Cette confrontation consisterait en la comparaison des valeurs produites par le code généré et celles obtenues depuis CMF. Une telle fonctionnalité permettrait de sécuriser le processus industriel de production de nouveaux modèles par ITK. Enfin, nous pensons intégrer à l'étape de génération de code la génération d'une documentation donnant les informations clés du code produit comme la configuration retenue et les scenarii appliqués. Dans le cas où le modèle continue à évoluer après une première version exploitée en production, cela permettrait d'améliorer le suivi des versions du modèle mises en production.

A l'activité de vérification est souvent associée la validation. Celle-là a pour objectif de vérifier que les modèles sont conformes à un ensemble de règles associées au DSML. Certaines de ces règles sont d'ores et déjà établies pour CMF : un modèle composite ne peut être vide, une entrée d'un modèle exécutable doit être alimentée par une source d'informations. Nous n'avons pas encore listé l'ensemble des règles à appliquer dans notre contexte. Elles seront fort probablement mises en œuvre à l'aide du cadriciel de validation associé à EMF[29]. La définition de la liste de règles à appliquer sera menée en concertation avec les agronomes. En effet, le comportement à adopter suivant les cas de figure devra être adapté à leurs besoins. La validation du modèle peut très bien être envisagée par la mise en place :

- d'erreurs rendant impossibles l'exécution et la génération de code ;
- d'avertissements que le modélisateur pourra ignorer ;
- de propositions de solutions alternatives : par exemple, une entrée *E* d'un modèle M_1 pourrait être reliée à une sortie *S* d'un modèle M_2. Le modèle M_1 étant exécuté avant le modèle M_2. Cette construction pourrait être interdite ou alors autorisée à condition qu'une valeur par défaut soit définie pour *S*.

[29] http://www.eclipse.org/modeling/emf/?project=validation#validation

Définir les règles de vérification en concertation avec les modélisateurs nous paraît indispensable afin d'obtenir un environnement qui les assiste dans la conception de modèles sans se montrer trop restrictif.

4.5- Réutilisation

Parmi les fonctionnalités que nous souhaiterions intégrées à CMF, la réutilisation de modèles devrait permettre d'améliorer la productivité des modélisateurs agronomes. Certains formalismes, comme la loi de Beer-Lambert déterminant l'interception lumineuse, sont couramment employés dans de multiples modèles grande culture. Permettre au modélisateur d'exporter un modèle exécutable ou un modèle composite entier afin de pourvoir le réutiliser dans un modèle futur est donc un atout. Ce type de fonctionnalités a été intégré à la plateforme Record (Bergez et al. 2013) par la mise en place d'une bibliothèque de modèles avec, pour chacun d'entre eux, une description complète du modèle, de ses entrées et de ses sorties. A l'instar de Record, nous souhaitons offrir au modélisateur la possibilité d'utiliser une base de données dans laquelle différents formalismes sont intégrés. Importer (exporter) des certains processus depuis (vers) cette base fait appel aux techniques de découpage de modèles (*slicing techniques*). (Blouin *et al.* 2011) définissent les *model slicers* comme « tools that extract a subset from a model, for a specific purpose". Dans notre cas, l'objectif est de procéder à l'enregistrement en base d'une partie du modèle pour permettre sa réutilisation ultérieure. Nous n'avons pas identifié, pour le moment, d'outillage dédié à ce type d'activité. Il semble réalisable d'implémenter ce comportement par exploration du fichier XMI généré depuis nos éditeurs et d'enregistrer en base les nœuds XML correspondant à la partie du modèle que le modélisateur souhaite exporter.

Le schéma de la base permettant cette persistance est encore à définir. Il paraît important qu'il prenne en compte la catégorisation des différents modèles suivant les processus biophysiques qu'ils représentent. Ainsi, l'ensemble des modèles sauvegardés formerait une base de connaissances utilisable par les modélisateurs. L'établissement de cette base passe par la mise en œuvre de ce que nous nommions « approche ontologique » au chapitre 3 (partie 1.3).

L'approche par métamodélisation et l'approche ontologique sont donc complémentaires pour obtenir un environnement abouti. L'avantage de notre démarche est que l'ontologie permettra de catégoriser les différentes représentations sans imposer de contrainte sur les formalismes utilisés, par opposition avec les approches par composants. De plus, l'import de ces formalismes sera réalisé avec des adaptations minimales :

- par l'emploi d'adaptateurs, ce qui n'aura aucun impact sur les modèles importés ;
- par modification des interactions avec le blackboard.

Ce dernier point est sans doute le plus gênant. En effet, il reste une adhérence forte entre la logique des modèles et la représentation du système plante-sol. Il sera peut-être possible d'y remédier par la mise en œuvre d'un mécanisme permettant de mettre en correspondance des variables d'état fournies par le *blackboard* avec celles attendues par le modèle réutilisé. Par contre, s'agissant des modèles donnant une dynamique à la structure même du *blackboard* il est peu probable de trouver une solution équivalente.

4.6- Autres évolutions envisagées

D'autres évolutions sont envisagées pour CMF, suivant les cas, elles ont pour vocation d'améliorer :

- l'expérience utilisateur ;
- la productivité ;
- le processus industriel de production de nouveaux modèles.

Nous avons évoqué ci-avant les aspects de vérification et de validation des modèles d'un point vue IDM. D'un point de vue modélisation et simulation, la validation est une étape importante de la conception d'un modèle scientifique mais dont les finalités sont différentes. Elle a pour objectif de confronter les résultats de simulation avec des données relevées sur les terrains. A terme, enrichir CMF d'une interface permettant de comparer des résultats de simulation avec des jeux de données par l'utilisation de fonctions statistiques serait un atout non négligeable. De plus, parmi les activités d'évaluation du modèle grande culture, certaines devraient être déléguées à des outils dédiés. Par exemple, la conduite d'analyse de sensibilité présente un grand intérêt pour le modélisateur. Elle permet d'évaluer la sensibilité du modèle construit à des modifications de ses paramètres et de ses données d'entrées. Ce type d'évaluation offre la possibilité de réduire le nombre de paramètres à prendre en compte, de connaître la précision nécessaire pour l'estimation de ces paramètres ou des données d'entrée.

La réalisation de ce type d'analyse passe par la définition d'un plan expérimental et, suivant le plan défini, le lancement de plusieurs milliers de simulation. Nous ne pensons pas ajouter ce type de fonctionnalité à CMF par contre il pourrait être intéressant d'y intégrer un interfaçage avec la plateforme OpenMOLE[30] (Open MOdeL Experiment) (Reuillon *et al.* 2010) conçue pour mener de telles expérimentations. Pour cela, nous imaginons une interface permettant de livrer le simulateur généré dans la plateforme. L'utilisation du simulateur par OpenMOLE pourrait requérir certaines adaptations, notamment pour permettre la variation des paramètres en cours de simulation dans le cadre d'une analyse de sensibilité.

Concernant les paramètres, une adaptation du métamodèle sera nécessaire. En effet, pour le moment un paramètre est associé à un unique modèle exécutable or, dans certains modèles grande culture, un même paramètre peut être utilisé par différents processus. En l'état, le modélisateur serait obligé de spécifier la valeur de ce paramètre autant de fois qu'il est utilisé. Il serait préférable de permettre une déclaration unique de cette valeur lorsque le modélisateur souhaite exécuter une simulation.

Nous souhaitons proposer aux modélisateurs une autre évolution de CMF ayant un impact à la fois sur les syntaxes abstraite et concrète mais aussi sur la sémantique du langage. Cette évolution se rapporte à la prise en compte de la phénologie. Le modèle phénologique reste en effet une particularité, il ne produit pas d'information à chaque pas de temps mais uniquement lors du changement de stade phénologique. Le plus souvent, le reste du modèle grande culture utilise le stade de développement de la plante comme un repère pour la définition de son comportement. Dans des cas plus rares, certains calculs sont déclenchés lorsqu'un stade phénologique donné a été atteint et ne sont utilisés qu'une seule fois. Ce type de calculs peut servir, par exemple, à modifier la valeur de certains paramètres de modèles exécutables. Tel que nous avons défini CMF et la syntaxe textuelle, ce type de fonctionnement est réalisable au niveau de chaque modèle par l'emploi d'une expression conditionnelle comparant la date actuelle de la simulation avec la date d'apparition du stade phénologique. Ce traitement n'apparaît pas optimal, en effet, cela signifie qu'un test est exécuté à chaque itération pour détecter un événement unique au cours de la simulation. Nous aimerions donc doter CMF d'une fonctionnalité conçue pour associer à un modèle exécutable une fonction qui serait déclenchée lors du passage à un stade phénologique donné. Cela consisterait donc à mettre en place une gestion événementielle dans le modèle grande culture. Une telle fonctionnalité

[30] http://www.openmole.org/

s'éloigne des habitudes de programmation des modélisateurs agronomes et se rapproche du domaine de la modélisation et de la simulation à événements discrets. Nous ne remettons pourtant pas en cause le raisonnement, établi au chapitre 2, dans lequel nous jugions les formalismes DEVS inadaptés à notre domaine. Ici, nous proposons d'ajouter un aspect événementiel sur un élément très spécifique du modèle grande culture, sans pour autant remettre en question la nature séquentielle à temps discret de nos modèles. Cela devrait améliorer la productivité des modélisateurs et surtout donner une meilleure visibilité sur l'existence de ces traitements particuliers au niveau du modèle conceptuel.

Pour améliorer encore l'expérience utilisateur, nous envisageons la mise en place d'éditeurs spécifiques dédiés à la mise en relation entre les fichiers de données météorologiques et les fournisseurs de données externes définis dans le simulateur. Au cours du prototypage du modèle scientifique, les données d'entrées nécessaires à la simulation proviennent systématiquement d'un ou de plusieurs fichiers météorologiques. Dans le cas où des données horaires et journalières sont utilisées, elles sont réparties dans deux fichiers distincts. Ce sont des fichiers textes, avec généralement un en-tête décrivant le contenu du fichier. La première ligne suivant l'en-tête définit le contenu de chaque colonne, le séparateur utilisé varie suivant le fichier : espace, tabulation, virgule ou point-virgule. Chaque ligne suivante correspond à une série de données pour une journée ou une heure. Notons que parmi les données figure la date (si besoin à l'heure près) à laquelle elles correspondent. Plutôt que d'exposer des classes techniques de lecture et de manipulation de fichiers textes, nous préférerions proposer une interface dont une ébauche est présentée par la figure 4.4.5.

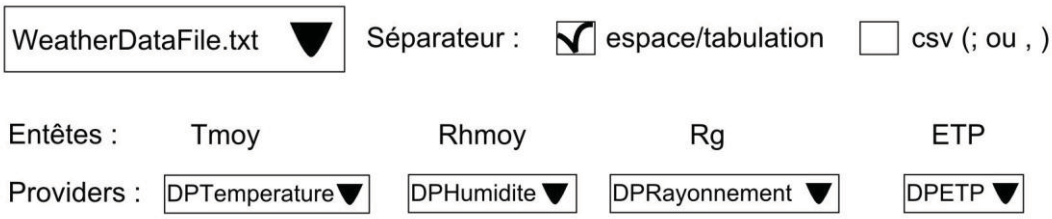


Figure 4.4.5 : Editeur associant les différents champs d'un fichier de données météorologiques aux fournisseurs de données externes du simulateur

A partir de cette interface il serait donc possible de sélectionner le fichier source d'informations météorologiques. Une fois que le type du séparateur utilisé dans ce fichier a été spécifié, les entêtes de colonne sont automatiquement chargés. Il reste alors à les associer aux fournisseurs de données externes appropriés. L'ergonomie de cette association est à étudier, elle pourrait se limiter à une sélection dans une liste déroulante ou à l'utilisation de glisser-déposer depuis la liste des entêtes vers celle des fournisseurs de données externes ou inversement. Outre le fait que ce système devrait faciliter la tâche des modélisateurs, plusieurs autres points d'intérêt peuvent se dégager. Tout d'abord, les dates présentes dans le fichier météorologique vont permettre de délimiter la plage de dates pendant laquelle la simulation est réalisable. Ainsi, avant le lancement de la première simulation, les dates de début et fin de simulation seront pré-renseignées en fonction de celles présentes dans le fichier. De plus, il sera possible de procéder à une vérification des données contenues dans le fichier en s'assurant : de la continuité des dates, qu'il n'y a pas de manques du type une donnée non renseignée à une date précise, que le format des données respecte celui attendu par les fournisseurs de données. Enfin, la définition de scénarii de test du modèle Java généré passe par l'exploitation des mêmes données météorologiques. En utilisant ce système, il sera possible de générer les tests en incluant les fichiers de données et la liaison avec les dataproviders. Cela nous permettra d'avoir une gestion complètement intégrée de nos tests.

Toujours dans l'idée d'améliorer l'expérience utilisateur, nous souhaitons également définir des vues sur le modèle grande culture en cours d'édition. Ces vues auront pour objectif de fournir des informations synthétiques sur le modèle. Ainsi nous pourrions fournir une vue définissant la séquence de simulation en fonction du pas de temps. La séquence étant répartie entre les différents modèles composites, il pourrait devenir difficile pour le modélisateur de conserver une connaissance complète sur son ordonnancement sans recourir à une telle vue. Cela est d'autant plus vrai lors le modélisateur découvre ou redécouvre un modèle. De la même manière, nous souhaiterions fournir une vue comprenant l'ensemble des modèles exécutables mettant en évidence les flux d'informations entre eux. D'autres vues pourraient s'avérer utiles au modélisateur avec la pratique. Enfin, nous imaginons la mise en place d'une génération de documentation consultable en dehors de CMF. Cette documentation pourrait être générée sous un format XHTML (eXtensible HyperText Markup Language) à l'instar des transformations DEVS2XHTML proposées par (Touraille 2012). Outre son universalité, l'avantage de ce format par rapport à un document texte classique est qu'il n'est pas linéaire. C'est-à-dire que, depuis un modèle donné, il sera possible d'accéder aux modèles qui

produisent des informations qu'il utilise par ses entrées ou d'accéder à ses suiveurs ou prédécesseurs dans la séquence d'exécution.

Les perspectives que nous venons d'exposer sont nombreuses, variées dans leurs objectifs et dans les ressources humaines et techniques qu'elles nécessitent pour être mises en œuvre. La syntaxe concrète textuelle et l'exécutabilité sont des éléments indispensables à l'obtention d'un environnement ayant le potentiel de se substituer à l'utilisation de Matlab dans le cadre de la conception des prototypes de modèles grande culture chez ITK. Cependant, pour convaincre les modélisateurs de son intérêt, il est indispensable d'intégrer nombres des fonctionnalités qui conféreraient à CMF un avantage par rapport à l'emploi de Matlab. Nous sommes en effet convaincus que, dans ces conditions, CMF répondrait beaucoup mieux aux besoins des modélisateurs agronomes d'ITK.

5- Conclusion

Dans ce chapitre, nous avons présenté la mise en œuvre de la conceptualisation exposée au chapitre 3. La liaison des différentes composantes du DSML - sa syntaxe abstraite, qu'est le métamodèle C3M, sa syntaxe concrète graphique et sa sémantique, portée par le cadriciel - a été accomplie dans l'environnement Eclipse à l'aide d'EMF, GMF et de la solution Acceleo. Le prototype de notre fabrique de modèles grande culture, qui en résulte, a été testé en prenant comme cas les modèles de croissance du blé et de la vigne. Dans le cas du modèle de la vigne, il a été possible d'obtenir à partir d'un nombre restreint de concepts visuels (70) la majeure partie (85%) de l'implémentation Java de ce modèle. Ce code généré s'appuie sur le cadriciel que nous avons défini. La pratique a montré que ce code s'avère aisément adaptable par un binôme ingénieur logiciel / expert du domaine.

Les résultats de notre travail de thèse offrent d'ores et déjà un intérêt pour la société ITK, tant du point de vue de la productivité que de celui de la sécurisation du processus industriel de production de nouveaux modèles. Le succès de cette preuve de concept devrait montrer l'intérêt de poursuivre l'effort de production d'une version aboutie de notre fabrique de modèles. Dans cette optique, la première étape à franchir est l'obtention d'éditeurs textuels permettant de définir le comportement des modèles exécutables, afin de franchir la seconde qui en est indissociable : la simulation des modèles dans l'environnement CMF. A partir de là, les perspectives sont nombreuses. L'intégration d'une fonctionnalité permettant d'enrichir une base de connaissances et de favoriser la réutilisation de modèles semble un impondérable. Ensuite, fournir le moyen de tester tant le simulateur généré que le prototype du modèle par

l'intégration de tests statistiques, ou encore permettre la construction de plans expérimentaux pour explorer le modèle conçu seraient des avantages indéniables. Ils permettraient d'obtenir un outillage robuste couvrant l'ensemble du cycle de production de modèles de la société ITK. Le léger changement de paradigme que nous suggérons par la gestion événementielle des changements de stades phénologiques pourrait, une fois adopté, faciliter encore la mise en place des modèles par les experts du domaine. Enfin, l'obtention d'une documentation générée automatiquement à partir CMF devrait encore améliorer la découverte des modèles ainsi que le suivi du processus de production.

Conclusion générale

Le travail que nous venons de présenter se situe à la croisée des chemins entre agronomie, ingénierie logiciel, plus particulièrement l'ingénierie dirigée par les modèles (IDM), ainsi que la modélisation et la simulation (M&S) numérique. Nous avons souhaité appliquer les concepts de l'IDM et certaines techniques qui lui sont associées afin de répondre à une problématique de production de modèles grande culture dans le contexte industriel de la société ITK. En effet, le processus de production qui y est pratiqué conduit à la mise en place d'un prototype scientifique dans l'environnement Matlab® par des experts modélisateurs. L'absence d'outils adaptés pour la modélisation conceptuelle et la documentation du modèle résulte en ce que l'implémentation du prototype devienne la référence sur laquelle les ingénieurs en génie logiciel doivent s'appuyer pour obtenir une implémentation en Java du modèle exploitable en production. La mise en place de cette double implémentation a un coût en termes de productivité dont il serait possible de s'affranchir. De plus, compte tenu du temps nécessaire à l'obtention d'un prototype satisfaisant, la maintenance, en parallèle, des deux implémentations présente des risques de disjonction. Ce risque est encore renforcé par le manque de documentation relative au modèle.

Comme nous l'avons vu au chapitre 2, les différentes techniques issues de l'IDM offraient un potentiel intéressant pour répondre à la problématique que nous nous sommes fixée. Alors que les différentes solutions de modélisation et de simulation, que nous avons évaluée, étaient inadaptées. Parmi celles-là, les solutions de modélisation pour l'agronomie sont avant tout dédiées à la recherche académique et ne permettent pas répondre à une problématique de nature industrielle. Parmi les solutions issues d'initiatives conjointes entre l'industrie et la recherche académique, telles que Modelica, les fonctionnalités présentant un intérêt pratique

étaient contrebalancées par des formalismes inadaptés aux modèles couramment utilisés par ITK.

Dans ce cadre, nous avons proposé l'adoption d'une approche IDM pour la conception d'une fabrique de modèles grande culture : CMF (*Crop Model Factory*). Cette fabrique vise à la fois :

- à sécuriser le processus de production par l'emploi de techniques générant du code Java ;
- à gagner en productivité grâce à un langage dédié à la M&S, dans le domaine de la modélisation grande culture, plus concis qu'un langage généraliste. De plus, les techniques génératives permettent de limiter l'intervention des ingénieurs logiciel ;
- à améliorer la documentation relative aux modèles agronomiques aussi bien pour assister le modélisateur dans la (re)découverte d'un modèle que pour assurer le suivi du modèle utilisé en production.

Cette fabrique repose sur la mise en œuvre d'un DSML (Domain-Specific Modelling Language). Dans le cadre de notre travail de thèse, nous nous sommes focalisés sur la partie de ce langage permettant de définir : la séquence d'exécution du modèle, les flux d'informations et la hiérarchie du modèle. Ces éléments constituent le modèle conceptuel du modèle grande culture. Pour définir ce DSML, nous avons proposé, au chapitre 3, sa syntaxe abstraite sous la forme d'un métamodèle (C3M) et une série de représentations graphiques constituant sa syntaxe concrète. La sémantique de notre DSML trouve sa réalisation dans le cadriciel que nous avons conçu et détaillé au chapitre 3.

C'est à partir de ces éléments qu'a été produit le prototype de CMF. Nous avons présenté, au chapitre 4, l'environnement Eclipse et ses plugins orientés IDM : EMF, GMF, Acceleo dont l'utilisation a conduit à l'obtention du prototype. Ce prototype permet d'ores et déjà de définir :

- la décomposition du modèle grande culture en un ensemble de sous-processus ;
- les flux d'informations au niveau du modèle ;
- la séquence d'exécution.

Pour un modèle donné, cette définition est obtenue à partir d'un nombre restreint de concepts visuels. Elle présente l'avantage d'obtenir, par génération automatique de code, près de 85% de l'implémentation en Java du modèle grande culture. Elle assure particulièrement la production automatique de la partie du code qui présentait le plus de risque d'erreurs lors de la traduction du prototype Matlab du modèle scientifique vers le langage Java.

Par ailleurs, le cadriciel, que nous avons produit pour faciliter la génération de code, permet l'adaptation aisée d'un modèle Java avec l'intervention d'un binôme ingénieur logiciel / modélisateur. Cette intervention est réalisable en dehors du cadre de CMF, ce qui offre la possibilité d'adapter le code produit à partir de CMF. Ce mode d'intervention n'est pas celui que nous privilégions. Cependant, il permet de circonvenir à des défauts qui pourraient être associés au statut de prototype de CMF. De manière plus générale, malgré le soin apporté à notre conception, il est possible que certaines particularités n'aient pas été prises en compte dans notre démarche ; disposer d'un moyen simple et efficace d'adapter le code obtenu est donc essentiel. En tout état de cause, nous pensons que CMF offre une opportunité intéressante pour la société ITK de mieux maîtriser la production de modèles grande culture et de mieux valoriser le patrimoine qu'ils représentent.

Telle que nous la concevons dans sa version aboutie, CMF permettra de couvrir l'ensemble du cycle de production : de la conceptualisation du modèle jusqu'à l'obtention de l'implémentation utilisable dans les outils d'aide à la décision produits par ITK. Le prototype obtenu dans le cadre de notre travail de thèse se veut comme une preuve de concept. Certes, dans le cadre du prototypage, certaines technologies nécessaires à un aboutissement complet de notre fabrique n'ont pas été explorées faute de temps. Cependant, les solutions techniques potentielles que nous proposons en perspectives de ce mémoire nous paraissent crédibles pour l'obtention d'un environnement complet de M&S dédié aux modèles grande culture. Son obtention passe par la réalisation d'éditeurs textuels permettant de définir le comportement des modèles exécutables. A cette fonctionnalité doit être associée l'exécutabilité des modèles dans l'environnement CMF.

Cependant, pour remporter l'adhésion des modélisateurs, il paraît essentiel d'intégrer des fonctionnalités évoluées qui font sans doute défaut à l'environnement Matlab. Parmi celles-là, la construction d'une base de connaissances devrait conduire à une réutilisation aisée des modèles patrimoniaux. De plus, outre la conception de modèles grande culture, la tâche des modélisateurs consiste également à en assurer son évaluation. Celle-là revêt différentes

formes, il s'agit à la fois de valider le modèle construit par sa confrontation à des données de terrain, mais également à explorer le modèle par la réalisation d'analyses telles que l'analyse de sensibilité. Intégrer ce type d'activité dans CMF, par l'ajout de bibliothèques statistiques pour la validation et un interfaçage avec une plateforme telle qu'OpenMOLE pour son exploration, présente un avantage substantiel par rapport aux pratiques actuelles dans la société ITK.

De plus, adopter une gestion événementielle des changements de stade phénologique devrait faciliter à la fois la conception du modèle grande culture et sa lisibilité. Mis à part le modèle phénologique, il est important de noter que CMF ne propose pas de formalismes spécifiques à la biologie. Or il serait possible d'étendre le concept de modèle phénologique pour permettre la formalisation de machines à état. Couvrir un champ d'application plus large que le seul domaine de la modélisation et de la simulation de la croissance des plantes devient alors concevable. Cette possibilité pourrait être envisagée si CMF devait, comme nous l'espérons, remplir toutes ses promesses.

Enfin, en introduction de ce mémoire, nous avons présenté l'ingénierie dirigée par les modèles comme étant le futur de l'ingénierie logiciel. Notre travail s'est avéré un bon moyen d'évaluer les forces et les faiblesses de l'IDM. Fournir aux spécialistes d'un domaine la capacité de définir des modèles et d'en obtenir des programmes satisfaisants est un des objectifs majeurs de l'IDM. Force est de constater que cet objectif n'est, pour l'heure, que partiellement rempli. Dans le cadre de notre travail, nous avons relevé deux inconvénients majeurs.

Tout d'abord, le coût d'entrée dans le paradigme du « tout est modèle » est assez élevé. L'appropriation de la théorie et la navigation entre différents niveaux abstractions sont des défis intellectuels intéressants mais difficiles à relever. C'est à force de pratique que le concepteur d'un DSML gagne en aisance. A l'heure actuelle, nous découvrons encore de nouvelles façons de voir et de concevoir dans un environnement dirigé par les modèles.

Le deuxième inconvénient est beaucoup plus important. En effet, il pourrait être responsable de l'insuffisante adoption de l'IDM en ingénierie logiciel. La profusion d'outils permettant de mener des approches IDM peut désorienter le novice. Tous ces outils ne sont pas équivalents, pour certains ils sont dédiés à des tâches très précises. Le nouveau venu dans le domaine de l'IDM ne dispose pas, à l'initiation de sa démarche, des critères de choix déterminant le meilleur outil à adopter. Comment imaginer, en effet, que son souhait de permettre l'édition

d'un modèle de son domaine, à partir de plusieurs éditeurs spécifiques, en fasse partie ? Les interfaces applicatives riches en fonctionnalités sont, de nos jours, très répandues pourtant tous les environnements orientés IDM ne les permettent pas. De plus, pour le découvrir, ce nouveau venu n'aura d'autre choix que la pratique, les documentations offertes pour les différents outils étant bien souvent trop rares. Cela rehausse encore le coût d'entrée dans l'IDM et menace ses chances d'adoption par un large public d'ingénieurs logiciel. Cette menace est renforcée par la difficulté d'interfaçage entre différents outils conçus pour des démarches orientées IDM, par exemple, la liaison entre des éditeurs GMF et des éditeurs Xtext que nous avons abordée au chapitre précédent. Ces interfaçages doivent être réalisés de manière artisanale alors que l'on pourrait espérer une intégration complète de ces outils au sein d'un même environnement.

Pour autant, nous continuons de penser que l'IDM représente l'avenir de l'ingénierie logiciel. Ce domaine reste, en effet, relativement jeune et, au regard de sa complexité, il n'est pas étonnant qu'une dizaine d'années n'aient pas suffi à obtenir des environnements complètement mûrs. Nous espérons que les possibilités théoriques et pratiques offertes par l'IDM permettront de convaincre la communauté et de stimuler l'essor technologique nécessaire à l'atteinte de sa pleine maturité.

Références

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludascher and S. Mock (2004). "Kepler: an extensible system for design and execution of scientific workflows". *Proceedings of 16th Conference on Scientific and Statistical Database Management*(Eds), 423-424.

Amthor, J. S. (2000). "The McCree - de Wit- Penning de Vries - Thornley respiration paradigms: 30 years later." *Annals of Botany*, **86**: 1-20.

Arango, G. (1989). "Domain analysis: from art form to engineering discipline." *SIGSOFT Software Engineering Notes*, **14**(3): 152-159.

Assmann, U., S. Zschaler and G. Wagner (2006). "Ontologies, Meta-models, and the Model-Driven Paradigm". *In Ontologies for Software Engineering and Software Technology*. C. Calero, F. Ruiz and M. Piattini, Springer**:** 249-273.

Atkinson, C. and T. Kühne (2003). "Model-Driven Development: a metamodeling foundation." *Software*, **20**(5): 36-41.

Baggiolini, M. (1952). "Les stades repères dans le développement annuel de la vigne et leur utilisation pratique." *Revue romande d'Agriculture et d'Arboriculture*, **8**(1): 4-6.

Baker, D. N., J. R. Lambert and J. M. McKinion (1986). "Gossym : a simulator of cotton growth and yield." *Technical Bulletin of the S.C. Agricultural Experiment Station*, **1089**: 1-134.

Baker, D. N. and R. E. Meyer (1966). "Influence of stand geometry on light interception and net photosynthesis in cotton." *Crop Science*, **6**: 15-19.

Balci, O., J. D. Arthur and R. E. Nance (2008). "Accomplishing Reuse with a Simulation Conceptual Model". *2008 Winter Simulation Conference*, Miami, FL, S. J. Mason, R. R. Hill, L. Mönchet al (Eds), 959-965.

Barbier, G., V. Cucchi, F. Pinet and D. R. C. Hill (2013a - In press). "CMF: a Crop Model Factory to Improve Scientific Models Development Process". *In Progressions and Innovations in Model-Driven Software Engineering*, IGI Global**:** 16p.

Barbier, G., V. Cucchi, F. Pinet and D. R. C. Hill (2013b - In press). "Domain-Specific Modeling for a Crop Model Factory." *International Journal of Agricultural and Environmental Information Systems*: 13p.

Barbier, G., J. Flusin, V. Cucchi, F. Pinet and D. R. C. Hill (2012). "Vine Model Design using a Domain-Specific Modeling Language. Prototype and Proof of Concept". *Proceedings of the ESM 2012 European Simulation and Modelling Conference*, Essen, Germany, M. Klumpp, 100-106.

Barbier, G., F. Pinet and D. R. C. Hill (2011). "MDE in Action: First Steps towards a Crop Model Factory". *Proceedings of the ESM 2011 European Simulation and Modeling Conference*, Guimarães, Portugal, 130-137.

Barros, F. J. (1995). "Dynamic Structure Discrete Event System Specification: a New Formalism for Dynamic Structure Modeling and Simulation". *Proceedings of the 1995 Winter Simulation Conference*, Arlington, VA, C. Alexopoulos, K. Kang, W. R. Lilegdon and D. Goldsman (Eds), 781-785.

Beck, H., K. Morgan, Y. Jung, S. Grunwald, H.-y. Kwon and J. Wu (2010). "Ontology-based simulation in agricultural systems modeling." *Agricultural Systems*, **103**(7): 463-477.

Bergez, J.-E., P. Chabrier, C. Gary, M. H. Jeuffroy, D. Makowski, G. Quesnel, E. Ramat, H. Raynal, N. Rousse, D. Wallach, P. Debaeke, P. Durand, M. Duru, J. Dury, P. Faverdin, C. Gascuel-Odoux and F. Garcia (2013). "An Open Platform to Build, Evaluate and Simulate Integrated Models of Farming and Agro-Ecosystems." *Environmental Modelling & Software*, **39**: 39-49.

Bézivin, J. (2004). "In Search of Basic Principle for Model Driven Engineering." *CEPIS, UPGRADE, The European Journal for the Informatics Professional*, **5**: 27-33.

Bézivin, J. (2005a). "Model-Driven Engineering: An Emerging Technical Space". *International Summer School, GTTSE*, Braga, Portugal, R. Lämel, J. Saraiva and J. Visser (Eds), 36-64.

Bézivin, J. (2005b). "On the Unification Power of Models." *Software and systems modeling*, **4**(2): 171-188.

Bézivin, J., M. Didonet Del Fabro, F. Jouault and P. Valduriez (2005). "Combining Preoccupations with Models". *ECOOP 2005 First Workshop on Models and Aspects - Handling Crosscutting Concerns in Model-Driven Software Development* Glasgow, UK,

Bézivin, J. and O. Gerbé (2001). "Towards a Precise Definition of the OMG/MDA Framework". *Proceedings of the 16th IEEE international conference on Automated software engineering (ASE 2001)*, San Diego, USA, 273-280.

Blouin, A., B. Combemale, B. Baudry and O. Beaudoux (2011). "Modeling Model Slicers". *MODELS 2011*, J. Whittle, T. Clark and T. Kühne (Eds), 62-76.

Booch, G., A. Brown, S. Iyengar, J. Rumbaugh and B. Selic (2004). An MDA Manifesto. *Business Process Trends/MDA Journal,* 9 p Retrieved december 2012 from http://www.bptrends.com/publications.cfm

Bouman, B. A. M., H. Van Keulen, H. H. van Laar and R. Rabbinge (1996). "The 'School of de Wit' crop growth simulation models: a pedigree and historical review." *Agricultural Systems*, **52**(2/3): 171-198.

Brisson, N., M. Launay and N. Beaudoin (2009). *Conceptual basis, formalisations and parameterization of the STICS crop model*. 297 p.

Brisson, N. and A. Perrier (1991). "A semiempirical model of bare soil evaporation for crop simulation models." *Water Resources Research*, **27**(5): 719-727.

Brisson, N., D. Ripoche, M. H. Jeuffroy, F. Ruget, B. Nicoullaud, P. Gate, F. Devienne-Barret, R. Antoniotelli, C. Durr, G. Richard, N. Beaudoin, S. Recous, X. Tayot, D. Plenet, P. Cellier, J. M. Machet, J. M. Meynard and R. Delecolle (1998). "STICS: a Generic Model for the Simulation of Crops and their Water and Nitrogen Balance." *Agronomie*, **18**: 311-346.

Brisson, N., J. Wery and K. Boote (2006). "Fundamental concepts of crop models illustrated by a comparative approach". *In Working with dynamic crop models*. D. Wallach, D. Makowski and J. W. Jones, Elsevier**:** 261-284.

Brooks, C., E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao and H. Zheng (2008a). Heterogeneous Concurrent Modeling and Design in Java - Chapter 6 Model Transformation, Electrical Engineering and Computer Sciences University of California at Berkeley**:** 135-146.

Brooks, C., E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao and H. Zheng (2008b). Heterogeneous Concurrent Modeling and Design in Java - Chapter 8 Code Generation, Electrical Engineering and Computer Sciences University of California at Berkeley**:** 171-194.

Chikofsky, E. J. and J. H. Cross (1990). "Reverse Engineering and Design Recovery: a Taxonomy." *IEEE Software*, **7**(1): 13-17.

Chopard, J., C. Pradal, D. Barbeau, T. Cokelaer and C. Godin (2011). "Scientific Workflow for Reusing Plant/FSPM Models". *MODSIM2011 19th International Congress on Modelling and Simulation*, Perth, Australia, F. Chan, D. Marinova and R. S. Anderssen (Eds), 968-974.

Chow, A. C. H. and B. P. Zeigler (1994). "Parallel DEVS a Parallel, Hierarchical, Modular Modeling Formalism". *Proceedings of the 1994 Winter Simulation Conference*, Orlando, FL, USA(Eds), 716-722.

Conrad, M. (2009). "Testing-based translation validation of generated code in the context of IEC 61508." *Formal Methods in System Design*, **35**(3): 389-401.

Cuadrado, J. S., J. deLara and E. Guerra (2012). "Bottom-Up Meta-Modelling: An Interactive Approach". *In Proceedings of the MODELS 2012 Conference*. R. France, J. Kazmeier, R. Breu and C. Atkinson, Springer. **7590:** 3-19.

Curcin, V. and M. Ghanem (2008). "Scientific Workflow Systems -can one size fit all ?" *Proccedings of the IEEE, CIBEC'08*(Eds),

Dahl, O.-J. and K. Nygaard (1966). "SIMULA: an ALGOL-based simulation language." *Communications of the ACM*, **9**(9): 671-678.

Dalle, O., J. Ribault and J. Himmelspach (2009). "Design Considerations for M&S Software". *Proceedings of the WSC 2009 Winter Simulation Conference*, Austin, USA, M. D. Rosseti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, 944-955.

de Wit, C. T., R. Brouwer and F. W. T. Penning de Vries (1970). "The simulation of photosynthetic systems". *IBP/PP Technical meeting Trebon*(Eds), 47-50.

Deelman, E., G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz (2005). "Pegasus: a Framework for Mapping Complex Scientifc Workflows onto Distributed Systems." *Scientific Programming Journal*, **13**(3): 219-237.

deLara, J. and H. Vangheluwe (2002a). "AToM3: a Tool for Multi-Formalism and Meta-modelling". *In Fundamental Approaches to Software Engineering*. R.-D. Kutsche and H. Weber, Springer**:** 174-188.

deLara, J. and H. Vangheluwe (2002b). "Using Meta-Modelling and Graph Grammars to Process GPSS Models". *16th European Simulation Multiconference on Modelling and Simulation 2002*(Eds), 100-107.

DenHaan, J. (2008). "8 Reasons Why Model-Driven Aproaches (will) Fail". Retrieved September from http://www.infoq.com/articles/8-reasons-why-MDE-fails

Drouet, J.-L. and L. Pagès (2007). "GRAAL : growth, architecture, allocation". *In Functional-structural plant modelling in crop production*. W. U. F. Series. Wageningen. **22:** 165-174.

Duchaine, F., T. Morel and L. Y. M. Gicquel (2009). "Computational-Fluid-Dynamics-Based Kriging Optimization Tool for Aeronautical Combustion Chambers." *AIAA Journal*, **47**(3): 631-645.

Duncan, W. G. (1973). "SIMCOT: a simulation of cotton growth and yield." *In Modelling the growth of trees*. C. Murphy, J. D. Hesketh and B. Strain**:** 115-118.

Eker, J., J. W. Janneck, E. A. Lee, J. Liu, X. LIiu, J. Ludvig, S. Neuendorffer, S. Sachs and Y. Xiong (2003). "Taming Heterogeneity - The Ptolemy Approach." *Proceedings of the IEEE*, **91**(1): 127-144.

Engelmore, R. and T. Morgan (1988). *Blackboard Systems*. Addison - Wesley. 602 p.

Estublier, J., G. Vega and A. D. Ionita (2005). "Composing Domain-Specific Languages for Wide-Scope Software Engineering Applications". *In Model Driven Engineering Languages and Systems*. L. Briand and C. Williams. Berlin, Springer Berlin Heidelberg. **3713:** 69-83.

Favre, J. M. (2004a). "Foundations of Meta-Pyramids: Languages and Metamodels - Episode II: Story of Thotus the Baboon". *post-proceedings of Dagsthul Seminar on Model Driven Reverse Engineering*(Eds), 1-28.

Favre, J. M. (2004b). "Foundations of Model (Driven) (Reverse) Engineering : Models -- Episode I: Stories of the Fidus Papyrus and of the Solarius". *Proceedings of Dagsthul Seminar on Model-Driven Reverse Engineering*(Eds), 1-30.

Favre, J. M. (2004c). "Towards a Basic Theory to Model Model-Driven Engineering". *3rd UML Workshop in Software Model Engineering, WiSME 2004*, 13.

Favre, J. M., J. Bézivin and I. Bull (2006a). "Evolution, rétro-ingénierie et IDM : du code aux modèles". *In L'ingénierie dirigée par les modèles - Au-delà du MDA*, Hermes science, Lavoisier édition**:** 185-215.

Favre, J. M., J. Estublier and M. Blay-Fornarino (2006b). "Introduction". *In L'ingénierie dirigée par les modèles - Au-delà du MDA*, Hermes science, Lavoisier édition**:** 17-34.

Favre, J. M. and T. NGuyen (2004). "Towards a Megamodel to Model Software Evolution Through Transformations." *Electronic Notes in Theoretical Computer Science*, **127**(3): 59-74.

Feddes, R. A., P. J. Kowalik and H. Zaradny (1978). *Simulation of field water use and crop yield. Simulation monographs.* University of Wageningen, Pudoc. 188 p.

Fisher, R. A. (1921). "Studies in crop variation. I. An examination of the yield of dressed grain from Broadbalk." *The Journal of Agricultural Science*, **11**(2): 107-135.

Fisher, R. A. and W. A. Mackenzie (1923). "Studies in crop variation. II. The manurial response of different potato varieties." *The Journal of Agricultural Science*, **13**(3): 311-320.

Flusin, J. (2012). Développement d'une interface graphique pour la conception de modèles agronomiques - Stage de 2ème année**:** 66.

Fong, C. (2000). Discrete-Time Dataflow Models for Visual Simulation in Ptolemy II. *Department of Electrical Engineering and Computer Sciences*. Berkeley, University of California. MSc**:** 46 p.

Fournier, C., C. Pradal, G. Louarn, D. Combes, J.-C. Soulié, D. Luquet, F. Boudon and M. Chelle (2010). "Building modular FSPM under OpenAlea: concepts and applications". *6th*

InternationalWorkshop on Functional-Structural Plant Models, University of California, Davis, 109-112.

Fritzson, P. (2010a). "Introduction to Modeling and Simulation". *In Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley & Sons**:** 3-18.

Fritzson, P. (2010b). "A Quick Tour of Modelica". *In Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, John Wiley & Sons, Inc.**:** 19-69.

Fritzson, P. and V. Engelson (1998). "Modelica - A Unified Object-Oriented Language for System Modeling and Simulation". *Proceedings of the 1998 European Conference on Object-Oriented Programming*, Brussels, Belgium, E. Jul (Eds), 67-90.

Gamma, E., R. Helm and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 403 p.

Gate, P. (1995). *Ecophysiologie du blé*. Lavoisier. 429 p.

Gauthier, L., C. Gary and H. Zekki (1999). "GPSF : a Generic and Object-Oriented Framework for Crop Simulation." *Ecological Modelling*, **116**(2-3): 253-268.

Gray, J., J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema and J. Sprinkle (2007). "Domain-Specific Modeling". *In Handbook of Dynamic System Modeling*. P. A. Fishwick, Chapman & Hall/CRC**:** 7.1-7.20.

Greenfield, J. and K. Short (2003). "Software Factories Assembling Applications with Patterns, Models, Frameworks and Tools." *OOPSLA'03*, Anaheim, Canada, 16-27.

Gronback, R. C. (2009). "Graphical Modeling Framework Runtime". *In Eclipse Modeling Project. A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley, Pearson Education, Inc.**:** 353-502.

Gruber, T. R. (1995). "Toward principles for the design of ontologies used for knowledge sharing." *International journal of human computer studies*, **43**: 907-928.

Hakojarvi, M., M. Hautala, J. Ahokas, T. Oksanen, T. Maksimow, A. Aspiala and A. Visala (2010). "Platform for simulation of automated crop production." *Agronomy Research*, **8**(1): 797-806.

Hammer, G. L., M. J. Kropff, T. R. Sinclair and J. R. Porter (2002). "Future contributions of crop modelling - from heuristics and supporting decision making to understanding genetic regulation and aiding crop improvement." *European Journal of Agronomy*, **18**: 15-31.

Heller, R., R. Esnault and C. Lance (2004a). *Physiologie Végétale 1. Nutrition*. Dunod. 323 p.

Heller, R., R. Esnault and C. Lance (2004b). *Physiologie Végétale 2. Développement*. Dunod. 366 p.

Hemel, Z., L. C. L. Kats and E. Visser (2008). "Code generation by model transformation: a case study in transformation modularity". *In Software and systems modeling*, Springer Berlin / Heidelberg. **9:** 183-198.

Hill, D. R. C. (2000). Contribution à la Modélisation des Systèmes Complexes : Application à la Simulation d'Ecosystèmes. Clermont-Ferrand, Université Blaise Pascal. Habilitation à Diriger les Recherches**:** 123 p.

Hill, D. R. C. (2009). "Réflexions sur la Modélisation et la Simulation". *In Simulation informatique au service des Sciences de la Vie*. D. R. C. Hill**:** 19-39.

Hillyer, C., J. Bolte, F. van Evert and A. Lamaker (2003). "The ModCom modular simulation system." *European Journal of Agronomy*, **18**: 333-343.

Ingalls, D., S. Wallace, Y.-Y. Chow, F. Ludolph and K. Doyle (1988). "Fabrik A Visual Programming Environment." *OOPSLA '88 Conference Proceedings, SIGPLAN Notices*, **23**(11): 176-190.

Jallas, E. (1998). Improved Model-Based Decision Support by Modeling Cotton Variability and using Evolutionary Algorithms. Mississippi, Mississippi State University. PhD**:** 239 p.

Jamieson, P. D. and M. A. Semenov (2000). "Modelling nitrogen uptake and redistribution in wheat." *Field crops research*, **68**: 21-29.

Jamieson, P. D., M. A. Semenov, I. R. Brooking and G. S. Francis (1998). "*Sirius*: a Mechanistic Model of Wheat Response to Environmental Variation." *European Journal of Agronomy*, **8**: 161-179.

Jones, J. w., G. Hoogenboom, C. H. Porter, K. Boote, W. D. Batchelor, L. A. Hunt, P. W. Wilkens, U. Singh, A. J. Gijsman and J. T. Ritchie (2003). "The DSSAT Cropping System Model." *European Journal of Agronomy*, **18**: 235-265.

Jones, J. w., B. A. Keating and C. H. Porter (2001). "Approaches to modular model development." *Agricultural Systems*, **70**: 421-443.

Jones, J. w., D. Makowski and D. Wallach (2006). "Introduction to Section 2". *In Working with dynamic crop models***:** 251-256.

Kelly, S. and R. Pohjonen (2009). "Worst Practices for Domain-Specific Modeling." *Software*, **26**(4): 22-29.

Kelly, S. and J.-P. Tolvanen (2008). *Domain-Specific Modeling: Enabling Full Code Generation.* Wiley - IEEE Computer Society Press. 448 p.

Kent, S. (2002). "Model Driven Engineering". *IFM '02 Proceedings of the Third International Conference on Integrated Formal Methods*(Eds), 286-298.

Kolovos, D. S., L. M. Rose, S. b. Abid, R. F. Paige, F. A. C. Polack and G. Botterweck (2010). "EuGENia: Taming EMF and GMF using Model Transformation". *Proceedings of the 13th International Conference, MODELS 2010, Model Driven Engineering Languages and Systems*, Oslo, Norway, Springer-Verlag, 211-225.

Kornecki, A. J. and S. Johri (2006). "Automatic Code Generation: Model-Code Semantic Consistency". *Proceedings of the 2006 International Conference on Software Engineering and Practice SERP'06*, Las Vegas, Nevada, USA, H. R. Arabnia and H. Reza, 191-197.

Kramer, J. (2007). "Is Abstraction the Key to Computing ?" *Communications of the ACM*, **50**(4): 36-42.

Kühne, T. (2006). "Matters of (Meta-) Modeling." *Journal on Software and Systems Modeling*, **5**(4): 369-385.

Kurtev, I., J. Bézivin and M. Aksit (2002). "Technological spaces: an initial appraisal". *CoopIS, DOA'2002 Federated conferences*, Irvine,

Lacy, L. W., W. Randolph, B. Harris, S. Youngblood, J. Sheehan, R. Might and M. Metz (2001). "Developing a Consensus Perspective on Conceptual Models for Simulation

Systems". *Proceedings of the 2001 Spring Simulation Interoperability Workshop*, Orlando, FL (Eds),

Lawless, C., M. A. Semenov and P. D. Jamieson (2005). "A wheat canopy model linking leaf area and phenology." *European Journal of Agronomy*, **22**: 19-32.

Lee, E. A. (2010). "Disciplined Heterogeneous Modeling". *In MODELS 2010, Lecture Notes in Computer Science* D. C. Petriu, N. Rouquette and O. Haugen, Springer - Verlag Berlin Heidelberg. **6395:** 273-287.

Lindenmayer, A. (1968). "Mathematical Models for Cellular Interactions in Development." *Journal of Theoretical Biology*, **18**: 300-315.

Loomis, R. S., R. Rabbinge and E. Ng (1979). "Explanatory models in crop physiology." *Annual review of plant physiology*, **30**: 339-367.

Louarn, G. (2009). Analyse et Modélisation de l'Organogénèse et de l'Architecture d'un Rameau de Vigne (*Vitis vinifiera L.*). Montpellier, Ecole nationale supérieure agronomique de Montpellier. PhD**:** 198 p.

Martin-Clouaire, R. and J.-P. Rellier (2009). "Modelling and Simulating Work Practices in Agriculture." *International Journal of Metadata, Semantics and Ontologies*, **4**(1/2): 42-53.

Mattson, S. E. and H. Elmqvist (1997). "Modelica - An international effort to design the next generation modeling language". *7th IFAC Symposium on Computer Aided Control Systems Design*, Gent, Belgium, 5.

McCown, R. L., G. L. Hammer, J. N. G. Hargreaves, D. P. Holzworth and D. M. Freebairn (1996). "APSIM: a Novel Software System for Model Development, Model Testing and Simulation in Agricultural Systems Research." *Agricultural Systems*, **50**: 255-271.

Mens, T. and P. Van Gorp (2006). "A Taxonomy of Model Transformation." *Electronic Notes in Theoretical Computer Science*, **152**: 125-142.

Merilinna, J. and J. Pärssinen (2010). "Verification and Validation in the Context of Domain-Specific Modelling". *DSM'10 10th Workshop on Domain-Specific Modelling*, Reno/Tahoe, Nevada, USA(Eds), 6.

Merilinna, J., O.-P. Puolitaival and J. Pärssinen (2008). "Towards Model-Based Testing of Domain-Specific Modelling Languages". *8th OOPSLA Workshop on Domain-Specific Modeling*, Nashville, USA, 6p.

Mernik, M., J. Heering and A. M. Sloane (2005). "When and How to Develop Domain-Specific Languages." *ACM Computing Surveys*, **37**(4): 316-344.

Minsky, M. (1965). "Matter, Mind and Models". *International Federation of Information Processing Congress*(Eds), 45-49.

Muller, P.-A., F. Fleurey and J.-M. Jézéquel (2005). "Weaving Executability into Object-Oriented Meta-Languages". *MODELS 2005*, Montego Bay, Jamaica, L. Briand and S. Kent (Eds), 264-278.

Mussa, M., S. Ouchani, W. Al Sammane and A. Hamou-Lhadj (2009). "A Survey of Model-Driven Testing Techniques". *9th International Conference on Quality Software*, Jeju, Korea(Eds), 167-172.

OMG (2001). "Model-Driven Architecture (MDA)". Retrieved december 2012 from http://www.omg.org/cgi-bin/doc?ormsc/01-07-01.pdf

OMG (2002). "OMG - Meta Object Facility (MOF) specification. v1.4". Retrieved december 2012 from http://www.omg.org/spec/MOF/1.4/PDF/

Pop, A. (2008). Integrated model-driven development environments for equation-based object-oriented languages. *Department of Computer and Information Science*. Linköping, Sweden, Linköping universitet. 263 p.

Pradal, C., S. Dufour-Kowalski, F. Boudon, C. Fournier and C. Godin (2008). "OpenAlea: A visual programming and component-based software for plant modeling." *Functional Plant Biology*, **35**(9 & 10): 751-760.

Pruzinkiewicz, P. and A. Lindenmayer (2004). *The Algorithmic Beauty of Plants*. Springer Verlag. 228 p.

Quesnel, G., R. Duboz and E. Ramat (2009). "The Virtual Laboratory Environment - An operational framework for multi-modelling, simulation and analysis of complex dynamical systems." *Simulation Modelling Practice and Theory*, **17**(4): 641-653.

Quesnel, G., E. Ramat, J.-C. Soulié, D. Duvivier and D. Raphael (2012). "Virtual Laboratory Environment : un Environnement de Multimodélisation et de Simulation des Systèmes Complexes." *Studia Informatica Universalis*, **10**(1): 205-234.

Reuillon, R., F. Chuffart, M. Leclaire, T. Faure, N. Dumoulin and D. R. C. Hill (2010). "Declarative Task Delegation in OpenMOLE". *Proceedings of the 2010 International Conference on High Performance Computing and Simulation IEEE*, Caen, France, W. W. Smari and J. P. McIntire (Eds), 55-62.

Richards, L. A. (1931). "Capillary Conduction of Liquids Through Porous Mediums." *Physics*, **1**(5): 318-333.

Riou, C., C. Valancogne and P. Pieri (1989). "Un modele simple d'interception du rayonnement solaire par la vigne. Vérification expérimentale." *Agronomie*, **9 (5)**: 441-450.

Ripoche, A., J.-P. Rellier, R. Martin-Clouaire, N. Paré, A. Biarnès and C. Gary (2011). "Modelling Adaptative Management of Intercropping in Vineyards to Satisfy Agronomic and Environmental Performances under Mediterranean Climate." *Environmental Modelling & Software*, **16**(12): 1467-1480.

Rizzoli, A. E., I. O. Athanasiadis and F. Villa (2007). "Delivering environmental knowledge: a semantic approach". *21st International Conference on Informatics for Environmental Protection: EnviroInfo 2007*(Eds), 43-50.

Rizzoli, A. E., M. Donatelli, I. O. Athanasiadis, F. Villa and D. Huber (2008). "Semantic Links in Integrated Modelling Frameworks." *Mathematics and Computer in Simulation*, **78**: 412-423.

Robinson, S. (2007). "Editorial. The Future's Bright the Future's... Conceptual Modelling for Simulation!" *Journal of Simulation*, **1**: 149-152.

Robinson, S. (2008). "Conceptual Modelling for Simulation Part I: Definition and Requirements." *Journal of the Operational Research Society*, **59**(3): 278-290.

Rugaber, S. and K. Stirewalt (2004). "Model-Driven Reverse Engineering." *IEEE Software*, **21**(4): 45-53.

Seidewitz, E. (2003). "What Models Mean." *IEEE Software*, **20**(5): 26-32.

Selic, B. (2003). "The Pragmatics of Model-Driven Development." *IEEE Software*, **20**(5): 19-25.

Selic, B. (2012). "What will it take ? A view on adoption of model-based methods in practice." *Software and systems modeling*: 1-14.

Shenoy, R., B. McKay and P. P. Mosterman (2007). "On Simulation of Simulink Models for Model-Based Design". *In Handbook of Dynamic System Modeling*. P. A. Fishwick, Chapman & Hall/CRC**:** 37-1 - 37-21.

Sinclair, T. R. and N. Seligman (2000). "Criteria for publishing papers on crop modeling." *Field crops research*, **68**: 165-172.

Soulié, J.-C. (2011). Présentation de DEVS et VLE pour la Réécriture du Modèle Ecomeristem. Les jeudis de l'AMAP.

Sprinkle, J. and G. Karsai (2004). "A Domain-Specific Visual Language for Domain Model Evolution." *Journal of Visual Languages and Computing*, **15**(3-4): 291-307.

Steinberg, D., F. Budinsky, M. Paternostro and E. Merks (2008). *EMF : Eclipse modeling framework 2nd edition*. Addison-Wesley Professional. 744 p.

Sun, Y., Z. Demirezen, M. Mernik, J. Gray and B. Bryant (2008). "Is My DSL a Modeling or Programming Language ?" *Proceedings of the 2nd International Workshop on Domain-Specific Program Development*(Eds), 5 p.

Touraille, L. (2012). Application of Model-Driven Engineering and Metaprogramming to DEVS Modeling & Simulation. *Ecole Doctorale des Sciences pour l'Ingénieur*. Clermont-Ferrand, Université Blaise Pascal. 310 p.

Touraille, L., M. K. Traoré and D. R. C. Hill (2011). "A Model-driven Software Environment for Modeling, Simulation and Analysis of Complex Systems". *Spring Simulation Multiconference - Symposium on Theory of Modeling and Simulation (TMS/DEVS)*, Boston, USA, 229-237.

Tukey, J. W. (1962). "The Future of Data Analysis." *The Annals of Mathematical Statistics*, **33**(1): 1-67.

Turing, A. M. (1952). "The Chemical Basis of Morphogenesis." *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences.*, **237**: 37-72.

van der Zee, D.-J., M. Pidd, A. Tolk, K. Kotiadis and A. A. Tako (2011). Education on Conceptual Modeling for Simulation - Beyond the Craft: A Summary of a Recent Expert Panel Discussion. *SCS M&S Magazine,* 93-102 Retrieved from

van Ittersum, M. K., P. A. Leffelaar, H. van Keulen, M. J. Kropff, L. Bastiaans and J. Goudriaan (2003). "On approaches and applications of the Wageningen crop models." *European Journal of Agronomy*, **18**(3-4): 201-234.

vanDeursen, A., P. Klint and J. Visser (2000). "Domain-Specific Languages: An Annotated Bibliography." *Newsletter ACM SIGPLAN Notices*, **35**(6): 26-36.

VanDeursen, A., E. Visser and J. Warmer (2007). "Model-Driven Software Evolution: A Research Agenda". *Proceedings of the CSMR Workshop on Model-Driven Software Evolution (MoDSE 2007)*, Amsterdam, The Netherlands, D. Tamzalit (Eds), 41-49.

Varenne, F. (2010a). "Chap.3 Statistique et "loi mathématique hypothétique" chez R.A. Fisher (1921-1922) ". *In Formaliser le vivant : Lois, Théories, Modèles ?* Hermann. Paris, France, Sté ACORT Europe**:** 39-61.

Varenne, F. (2010b). *Formaliser le vivant : Lois, Théories, Modèles ?* Sté ACORT Europe, Paris, France. 394 p.

Visser, E. (2005). "A Survey of Strategies in Rule-based Program Transformation Systems." *Journal of symbolic Computation*, **40**(1): 831-873.

Viswanathan, K., M. Shur, P. R. Spalart and M. Strelets (2008). "Flow and Noise Predictions for Single and Dual Stream Beveled Nozzles." *AIAA Journal*, **46**(3): 601-626.

Völter, M. (2011). "MD/DSL Best Practices. Update March 2011". Retrieved September 27, 2012 from http://www.voelter.de/publications/index.html

Vos, J., J. B. Evers, B.-S. G.H., B. Andrieu, M. Chelle and P. H. B. deVisser (2010). "Functional-Structural Plant Modelling: a New Versatile Tool in Crop Science." *Journal of Experimental Botany*, **61**(8): 2101-2115.

Vos, J., L. F. M. Marcelis and J. B. Evers (2007). "Functional-structural plant modelling in crop production : adding a dimension". *In Functional-structural plant modelling in crop production*. W. U. F. Series, Wageningen. **22:** 1-12.

Wallach, D. (2006). "The two forms of crop models." *In Working with dynamic crop models*. D. Wallach, D. Makowski and J. w. Jones**:** 3-9.

Wang, E. and T. Engel (2000). "SPASS: a generic process-oriented crop model with versatile windows interface." *Environmental Modelling & Software*, **15**: 179-188.

Weir, A. H., P. L. Bragg, J. R. Porter and J. H. Rayner (1984). "A winter wheat crop simulation model without water or nutrient limitations. ." *The Journal of Agricultural Science*, **102**: 371-382.

Whittle, J. and P. Jayaraman (2007). "MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation". *In Workshops and Symposia at MoDELS 2007*. H. Giese. Nashville, TN, USA, Springer Berlin Heildelberg**:** 16-27.

Williams, J. R., C. A. Jones and P. T. Dyke (1984). "A Modelling Approach to Determining the Relationship between Erosion and Soil Productivity." *Transactions of American Society of Agricultural Engineers*, **27**: 129-144.

Willocquet, L., J. Aubertot, S. Lebard, C. Robert, C. Lannou and S. Savary (2008). "Simulating multiple pest damage in varying winter wheat production situations." *Field crops research*, **107**(1): 12-28.

Zeigler, B. P. (1976). *Theory of modelling and simulation*. John Wiley, New York. 435 p.

Zeigler, B. P., H. Praehofer and T. G. Kim (2000). *Theory of Modeling and Simulation, Second Edition*. Academic Press, Inc., Orlando, FL. 510 p.

Annexe 1 : Glossaire agronomique

Agronomie : étude scientifique des relations entre les plantes cultivées, le milieu (sol, climat) et les techniques agricoles.

Appareil caulinaire : désigne la tige et les organes qui s'y développent.

Appareil racinaire : Ensemble des racines d'une plante. L'appareil racinaire assure l'alimentation en eau et éléments minéraux de la plante (azote, phosphore, potassium…) ainsi que l'ancrage de la plante dans le sol.

Biométrie : Etude statistique des dimensions et de la croissance des êtres vivants (Le Petit Larousse).

Bourgeon : Structure apparaissant à la surface des tiges. Contenant un amas de cellules indifférienciées (le méristème), elle est destinée à produire de nouveaux organes.

Bourgeon axillaire : Bourgeon situé à l'aisselle d'une feuille.

Chlorophylle : Pigment caractéristique des végétaux effectuant la photosynthèse.

Croissance potentielle : La croissance potentielle désigne la croissance de la plante dans le contexte d'une disponibilité parfaite en eau et en nutriments.

Débourrement : Phase caractéristique des plantes pluriannuelles à la sortie de l'hiver les bourgeons laissant apparaître la matière protectrice du bourgeon : la bourre.

Développement : Le développement d'un être vivant désigne l'ensemble des stades et changements physiologiques associés qu'il rencontre au cours de sa vie.

Epiphyte : Caractérise les plantes qui se servent d'autres plantes comme support sans pour autant se comporter en parasite de la plante support.

Hybridation : désigne le croisement (i.e. la reproduction) entre deux espèces ou deux variétés différentes d'une même espèce.

Méristème apical : Amas de cellules indifférienciés à l'extrêmité d'une tige ou d'une ramification.

Morphogénèse : La morphogénèse correspond à l'ensemble des processus permettant à un organe ou un autre vivant d'atteindre sa forme finale.

Organe de réserve : Organe destiné à mettre en réserve des carbohydrates, dans le monde végétal le plus souvent sous la forme d'amidon. Suivant les espèces, l'organe peut être une feuille, une racine ou encore une tige.

Pétiole : Partie allongée de la feuille permettant son insertion sur la tige. En interne l'organisation des tissus du pétiole est très proche de celle de la tige.

Photosynthèse : Processus bioénergétique qui permet d'utiliser l'énergie lumineuse pour réduire le carbone atmosphérique. Cette réduction conduit à la production de matière organique carbonée. C'est le processus très majoritaire de réduction du carbone qui est à l'origine de la plupart des écosystèmes.

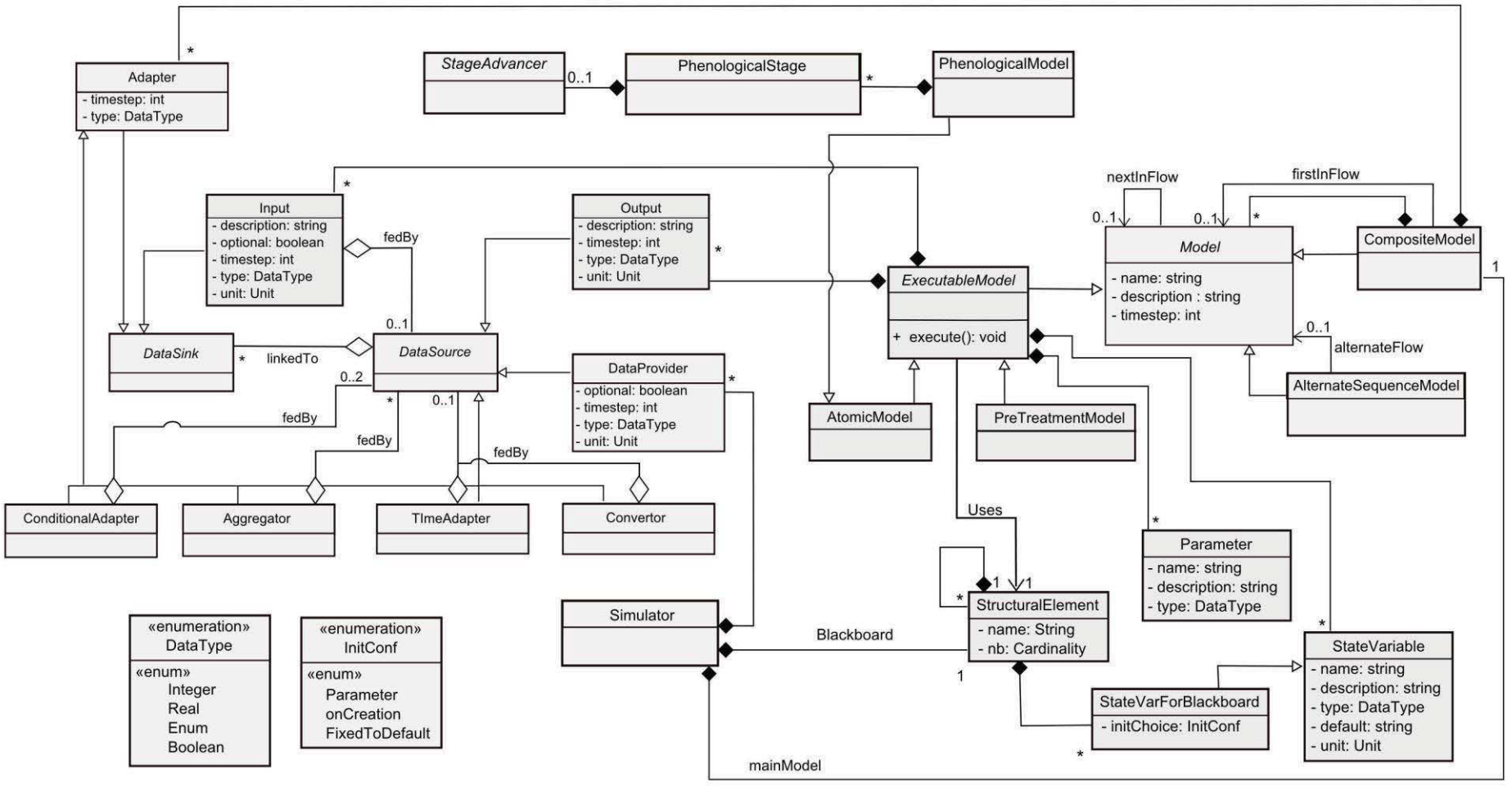
Phyllotaxie : Désigne l'étude de la disposition des feuilles et des ramifications des plantes.

Phytomère : Brique élémentaire d'une tige, il se caractérise à sa base par un nœud, porteur d'un bourgeon, suivi d'un entre-nœud à l'extrémité duquel est inséré une feuille

Structure reproductrice : Dans ce mémoire, regroupe toutes les structures intervenant dans la reproduction de la plante : fleurs, inflorescences, fruits, graines.

Végétal supérieur : (également appelé Embryophyte) toute plante à tige comportant un système vasculaire (par opposition aux mousses et aux fougères).

Annexe 2 : Le métamodèle C3M



Annexe 3 : Diagramme de classes du cadriciel