SILASOL :
un simulateur informatique de sole "grandes cultures" pour étudier la place du pois

Description du modèle et de l'outil

Jean-Pierre Rellier (INRA/MIA/UBIA) - Juin 2010

Plan

1. Introduction

SILASOL est **un ensemble de modules informatiques** qui permettent, de manière complémentaire, de modéliser la sole en grande culture d'une exploitation et de simuler le fonctionnement dynamique de ce système dans un contexte de contraintes et d'événements donné.
Il a été construit avec l'intention d'**étudier la place du pois dans les systèmes de culture**. Une hypothèse agronomique centrale est que l'innovation variétale peut faciliter l'insertion du pois, et donc développer son usage au bénéfice du système global.
Pour valider cette hypothèse, les indicateurs de réussite de l'insertion du pois (niveau et qualité des productions, bilan économique et environnemental, acceptabilité de l'organisation du travail) doivent être disponibles au niveau du système global, sur une longue période et dans une gamme de contextes. A cet effet, la méthode d'étude choisie est **la simulation virtuelle** fondée sur un modèle informatique du système, plutôt que le recueil de données par expérimentation ou enquête.
Une hypothèse agronomique complémentaire est que, pour être pertinente, la valeur des indicateurs de réussite doit résulter de l'interaction réaliste entre les phénomènes biophysiques naturels (les cultures dans leur environnement pédo-climatique) et les interventions humaines (l'installation et la conduite des cultures, dans leur contexte informationnel). Cette hypothèse oriente sur la **simulation du fonctionnement dynamique du système** et de ses interactions internes, situées dans le temps, plutôt que sur l'utilisation de variables synthétiques telles que des bilans sur périodes. C'est pourquoi SILASOL a été développé dans l'environnement logiciel DIESE, qui fournit des structures de données et de connaissances, des algorithmes, des services, pour exprimer ces interactions et pour calculer leurs conséquences. Les structures et les algorithmes sont rassemblés dans une **ontologie**[1], implémentée dans un **ensemble de classes d'objets programmées en C++**.

2. Architecture logicielle

Le modèle du système « sole de grandes cultures » est la réunion d'un ensemble de données et de connaissances disparates et d'origines diverses.
- les données sur le climat sont des enregistrements météo en une station représentative du contexte souhaité.
- les données sur les sols sont essentiellement hypothétiques.
- les connaissances sur le fonctionnement pois-sol-climat sont celles dans le simulateur informatique

[1] Martin-Clouaire, R. et Rellier, J.-P (2006) Fondements ontologiques des systèmes pilotés. Rapport interne UBIA-INRA, Toulouse-Auzeville. http://carlit.toulouse.inra.fr/diese, rubrique 'Publications'/'Rapports techniques'.

AFISOL[2], lui-même extension du modèle AFILA[3]. Elles sont donc très développées.
- les connaissances sur le fonctionnement des autres cultures sont beaucoup plus sommaires, en relation avec la focalisation de l'analyse sur le pois.
- sur le pois, l'important indicateur sur l'état de tassement du sol est calculé par le programme informatique SISOL[4].
- les connaissances sur l'organisation du travail sont en partie une expertise détenue par les modélisateurs, et en partie des hypothèses sur le contexte technique et les options d'organisation de l'agriculteur.

Cet ensemble de connaissances, spécifique du domaine étudié, est installé comme une couche reposant sur trois autres couches de concepts et, dans le simulateur informatique, de classes C++, qui sont sans référence à un domaine d'application particulier (figure 1). Dans la couche la plus basse, on parle d'objet, d'attribut, de méthode, de relation, de fonction et procédure : des concepts universels de la modélisation « objet ». Au-dessus, DIESE impose de décrire un système à l'aire de trois classes d'objets fondamentaux : les entités, les événements et les processus, et propose tous les services pour les créer et les manipuler (couche BASIC DIESE). Encore au-dessus, la couche CONTROL DIESE propose un ensemble de concepts par lesquels on modélise le pilotage (la conduite) du système : les activités, les opérations, les ressources, etc., et les services associés. La cible de SILASOL entraîne évidemment que cette couche-là est exploitée.

Pour être plus précis, la couche applicative est elle-même l'adjonction de deux sous-couches.
Il y a d'abord les connaissances relatives au domaine étudié : les modèles des cultures envisageables (bilans hydrique, azoté, développement et croissance, etc.), les modèles biophysiques du tassement du sol, de la sensibilité du pois au gel, etc., le modèle de la conduite (en termes de déroulé de plans d'activités, de raisonnement de l'allocation des ressources, de mise en œuvre d'ajustements, etc. ...).
Il y a ensuite, et c'est le dernier niveau, la représentation de l'instance de système dont on veut regarder le fonctionnement dynamique. Ce sont des données singulières à un cas d'étude (localisation climatique, caractéristiques du parcellaire, pool de ressources disponibles, choix des cultures et mode de pilotage adopté).

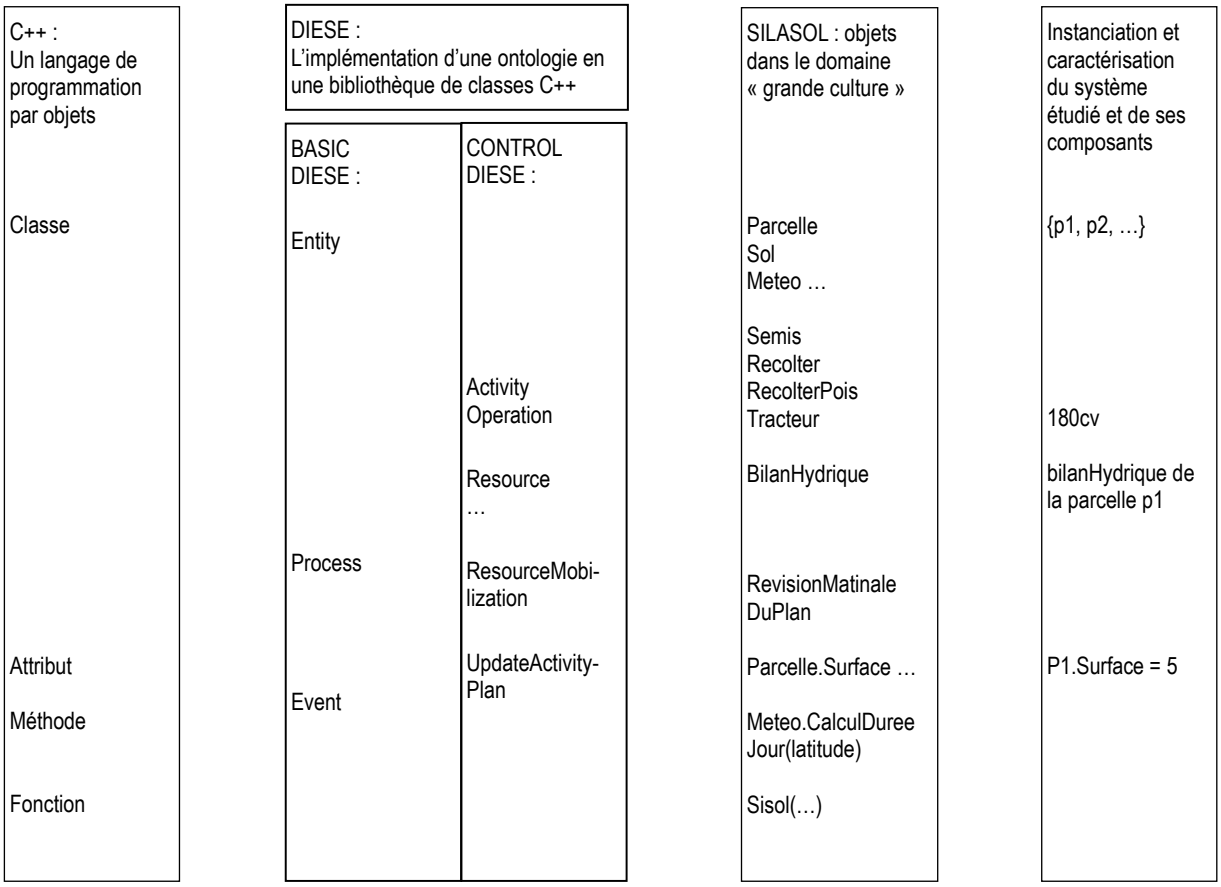


Figure 1 : Les couches de connaissances dans l'architecture logicielle de SILASOL

Les connaissances ainsi organisées sont traitées par un moteur de simulation, l'ensemble relevant de la notion de

2 Vocanson, A., 2006.
3 Biarnes, A. et al., 2004.
4 Roger-Estrade, R. et al., 2004.

« système à base de connaissance ». Le rôle du moteur, dans un simulateur de dynamique temporelle, est de synchroniser les changements d'état provoqués par les différents processus en jeu. Il exploite à cet effet deux types de connaissances exprimées par le développeur :

- les instances des classes d'événements (Event), caractérisées par leur date d'occurrence, et empilés par date croissante dans un « agenda ». Elles sont créées en préambule du lancement de la dynamique, ou bien en cours de simulation. L'agenda est « dépilé » de l'événement le plus immédiat, tant qu'il n'est pas vidé.
- les « pas » de progression des processus de nature continue (par opposition aux processus ponctuels). Par exemple, un processus dont le pas est de 1h devra être « recalculé » 24 fois quand un autre de pas journalier ne le sera qu'une fois.

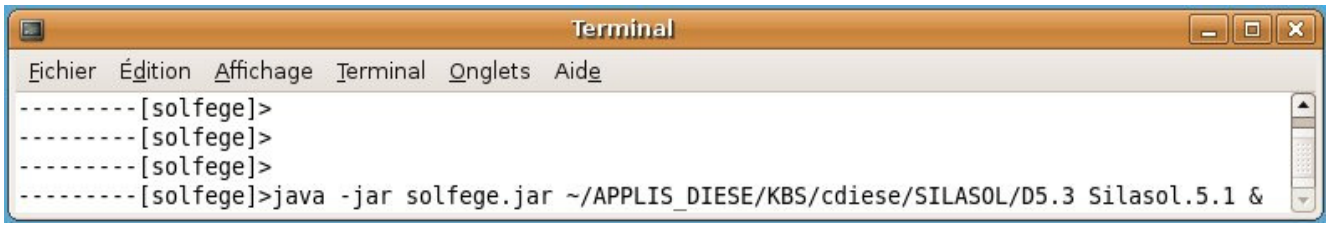
2. Interface et utilisation

Un simulateur développé sur DIESE est un système dont le moteur est fermé et la base de connaissance ouverte. Autrement dit, le développeur du simulateur (c'est-à-dire le programmeur du modèle) non seulement peut, mais doit ne s'occuper de la rédaction de la base de connaissance, en connaissance du mode de fonctionnement du moteur. En particulier, il ne programme pas l'avancée du temps.
Quant à base de connaissance, on peut la voir composée de deux ensembles :

- les connaissances qualifiées d'internalisées : c'est le développeur qui les a exprimées et qui les a traduites sous forme de classes d'objets, de méthodes, fonctions, etc., par le langage C++. Cette expression est informatiquement compilée en un programme exécutable sur une machine, et de ce fait non modifiable par l'utilisateur du simulateur. C'est la couche 'SILASOL …' de la figure 1. Elle est très généralement développée à l'aide de l'interface Solfege.
- les connaissances qualifiées d'externalisées : c'est l'utilisateur du simulateur qui les exprime, en donnant au moteur de simulation des directives d'instanciation des classes internalisées. Ces directives sont écrites, en un langage spécifique à DIESE, dans les fichiers (au format 'texte') dont les emplacements sont fournis en argument de la commande système qui lance le simulateur.

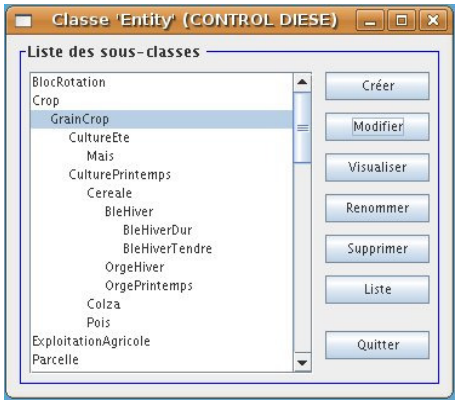
2.1 L'interface Solfege

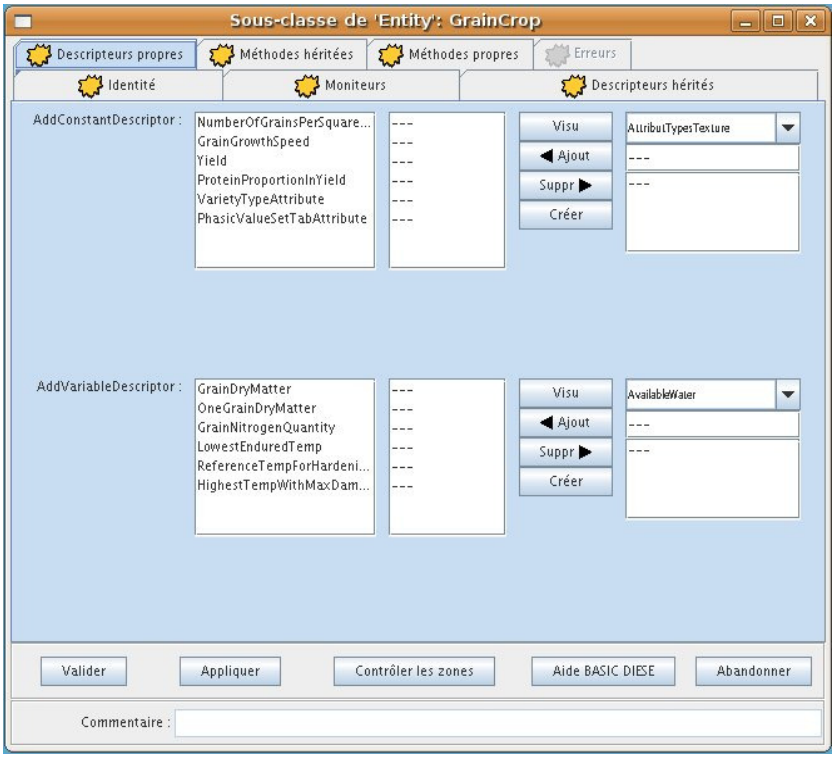
La documentation 'Utilisateur' est fournie avec la plateforme DIESE, dans un ensemble de pages HTML. L'interface est lancée par une ligne de commande dans un 'terminal' du système d'exploitation :



Les deux derniers arguments de la commande 'java' sont le nom du répertoire terminal contenant la base de connaissance, précédé de son chemin d'accès à partir du répertoire 'home' de l'utilisateur. Cette commande fait apparaître l' établi des outils de développement (menu 'Développement') et de génération du simulateur exécutable (menu 'Génération') :

A chaque classe d'objet correspond une fenêtre de développement spécifique, adaptée à sa conception dans la couche DIESE. A titre d'exemple, les deux images ci-dessous montrent la fenêtre de modification de la classe 'GrainCrop' (ouverte sur son onglet 'Descripteurs propres') qui apparaît par une action dans la fenêtre exhibant l'ensemble des classes d'entités.



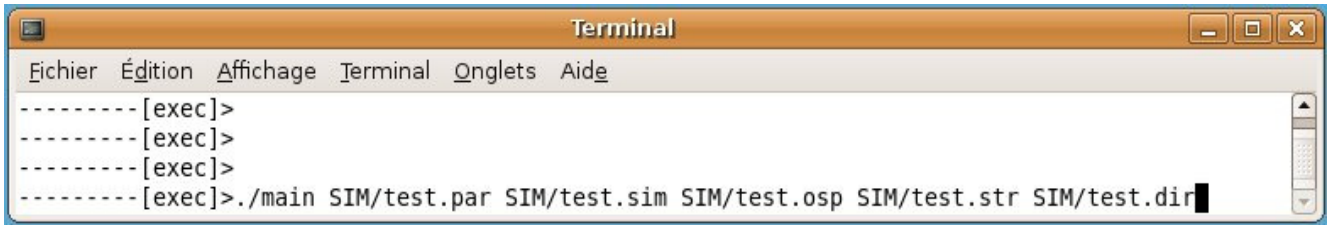


Ces fenêtres de développement permettent notamment de doter la classe de descripteurs (attributs) et de méthodes, et de préciser une éventuelle procédure à exécuter lors de chaque instanciation de la classe (par exemple instancier les différents horizons d'un sol lors de l'instanciation de ce sol).

La génération du simulateur exécutable est commandée par les items du menu 'Génération'. Elle comprend trois phases ! (i) la génération du code source C++ à partir des spécifications données dans les fenêtres de développement, (ii) la création, une fois pour toutes, d'un fichier de directives de compilation adapté au système hôte et à l'arborescence de fichiers propre à l'utilisateur, (iii) la génération de l'exécutable par la compilation du code source C++ et la phase d'édition des liens. Le simulateur est un programme exécutable de nom 'main', rangé dans le répertoire exec/ de la base de connaissance (voir l'arborescence des fichiers en Annexe).

2.2 Utilisation du simulateur

Bien que l'exploitation de l'interface Mi_Diese (aussi incluse dans la plateforme DIESE) soit possible, on présente ici l'autre mode d'utilisation du simulateur, celui en mode 'ligne de commande'. Il consiste à faire exécuter, dans un terminal du système hôte, une commande composée du nom du simulateur suivi d'une liste d'arguments. Cette liste est composée d'au moins 5 éléments, qui sont les emplacements de 5 fichiers de données et connaissances externalisées :



Dans l'image ci-dessus, le mot 'SIM est un lien vers un répertoire, dans lequel l'utilisateur a choisi de placer les 5 fichiers. Ces fichiers ont un nom libre, mais dans le jeu visualisé, ces noms ont un préfixe commun ('test') et ds suffixes qui évoquent leur contenu. Ce contenu est précisé dans le chapitre 5 'Connaissances externalisées', pour les suffixes 'par', 'sim' et 'str'.

Le fichier suffixé par 'dir' a, dans SILASOL, le contenu universel généré automatiquement par Solfege (menu

‘Livraison’). C’est lui qui contient la directive de lancement de la dynamique simulée du système, et c’est pourquoi il est placé en fin de liste, pour que ce lancement soit fait après la lecture des autres fichiers.

Le fichier de suffixe ‘osp’ permet de restreindre l’édition des résultats de simulation à ce qui est d’intérêt au moment du travail. On peut par exemple focaliser son analyse sur le sol, et à un autre moment sur les rendements des cultures. Voici un exemple de ‘spécification de sortie’, qui demande l’écriture dans le fichier ‘croissance.txt’ de toutes les valeurs de trois descripteurs des instances de la classe de nom ‘pois, avec un nombre de décimales approprié, et avec un format de date particulier :

```
SAVE DESCRIPTOR "pois"    ALL    "croissance.txt"    NEW    DATE_DMY
               "grossAboveGroundDryMatter"    1
               "netAboveGroundDryMatter"      1
               "leafAreaIndex"         3;
```

Le langage complet d’expression des spécifications de sortie est précisé dans la documentation HTML de DIESE, fournie avec la plateforme.

4. Les connaissances internalisées

Dans le cadre de ce document, l’énoncé des connaissances introduites dans la base SILASOL ne peut pas être exhaustif. Elles portent sur la structure du système étudié, son fonctionnement et sa dynamique, respectivement modélisés par des spécialisations des classes Entity, Process et Event de la couche BASIC DIESE.

4.1 La structure

La représentation de la structure du système est en Figure 2. L’entité de plus haut niveau d’intégration est l’exploitation agricole (➠ CD!ProductionSystem) dont le composant prédéfini CD!ControlledSystem est un SystemeBiophysique (➠ BD!Entity). Les deux composant de ce dernier sont un système de culture (➠ BD!Entity), lui-même ensemble de blocs-rotation (➠ BD!Entity), et une station météo (Weather ➠ BD!Entity). Un bloc-rotation est un ensemble de parcelles (➠ BD!Entity), chacune supportant la même rotation des cultures mais avec un décalage d’une campagne par rapport à sa « voisine » (une rotation de 4 cultures fera donc l’objet d’un bloc-rotation de 4 parcelles). Une parcelle est un ensemble de placettes (➠ BD!Entity). Il y a autant de placettes dans une parcelle que d’hectares de superficie. Les placettes vont supporter le caractère progressif des opérations techniques : l’opération est réalisée placette après placette, et en un instant donné, on peut identifier l’ensemble des placettes opérées. Si l’opération ne reprend que beaucoup plus tard, on pourra choisir de différencier les dynamiques d’état sur les deux parties ainsi identifiées de la parcelle. Parmi les placettes d’une parcelle, une seule sert de référence et de support des processus biophysiques, les autres étant supposées dans le même état et dans la même dynamique (tant qu’aucun événement n’invalide cette supposition). Les processus biophysiques portent sur les deux composants de la placette : la culture (➠ BD!Entity) et le sol (➠ BD!Entity). Un sol est un ensemble de couches de sol (SoilLayer ➠ BD!Entity), au nombre de une ou plus de une selon les réquisitions des modèles de culture (le modèle sol d’AFISOL porte sur un empilement de trois couches). Un descripteur notable d'une culture est son âge (PlantAge ➠ BD!Entity), lequel peut être exprimé sous diverses formes (nombre de jours depuis un repère tel que la levée, somme de degrés-jour depuis un repère, stade physiologique).

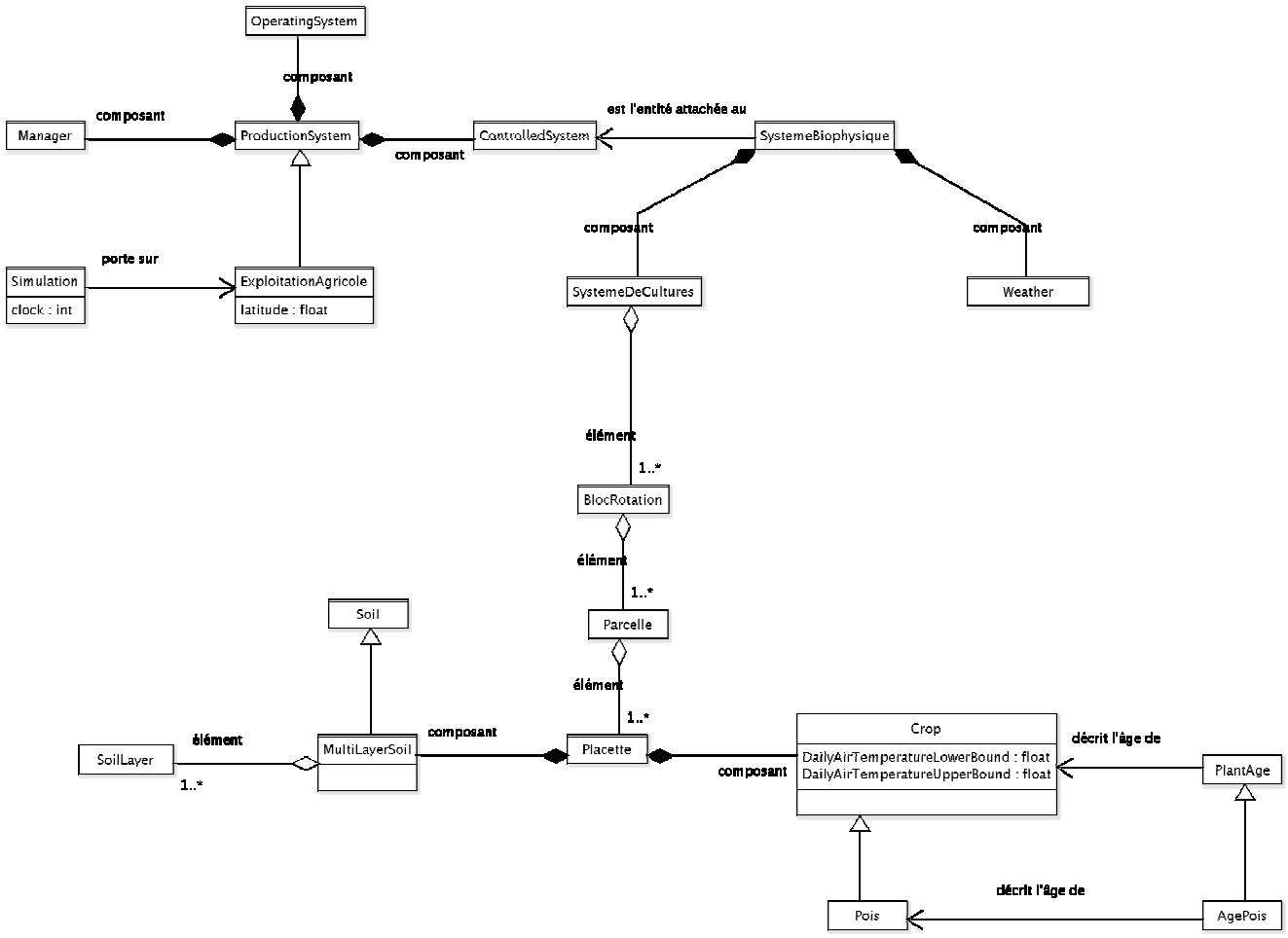


Figure 2 : la représentation simplifiée du système de production dans SILASOL.
Une flèche vide est une relation de spécialisation entre classes, un losange plein termine une relation de composition, une losange vide termine une relation ensembliste, une flèche simple exprime que deux entités entretiennent une relation d'un autre type.

4.1 Les processus du changement d'état

Les processus biophysiques en jeu dans le modèle de SILASOL sont passés en revue dans l'Annexe 1.

L'évolution du contexte météorologique du système est simulée par la lecture quotidienne d'un enregistrement dans un fichier de données météo journalières. Ce processus met à jour, au même rythme, les descripteurs d'une instance de la classe Weather, laquelle constitue une sorte de tableau noir librement consultable par les autres modules de connaissance.

Le processus d'évolution de l'âge des plantes est en partie commun à toutes les cultures, et en partie spécifique à chacune. Dans la partie commune, on augmente l'âge d'un jour et des degrés-jours de ce jour, puis on regarde si le moment est venu de changer de stade phénologique, sur la base de la définition de ces stades propre à chaque culture. Les stades sont définis par l'atteinte d'une date calendaire, ou du dépassement d'un nombre de degrés-jour accumulés, ou autrement encore. Dans l'éventuelle partie spécifique à une culture, on code des changements d'état qui lui sont propres. Par exemple, dans le développement du pois, on incrémente une somme de degrés-jour à partir du stade 'FSLA' qui ne concerne que cette culture.

La progression de l'enracinement n'est modélisée que pour le pois. Le changement d'état porte sur la progression du front racinaire, mais aussi sur les épaisseurs complémentaires des deux couches de sol sous l'horizon superficiel, information exploitée par le module de bilan hydrique. C'est l'opération de semis (SemerPois➠ CD!Operation) qui instancie, initie et lance le processus de progression de l'enracinement.

Le calcul de la satisfaction du besoin en eau du pois repose sur un modèle de bilan hydrique issu directement du modèle AFISOL. Le modèle met à jour les volumes d'eau dans les différentes couches de sol, calcule la quantité d'eau drainée au-delà des racines et met à jour la valeur de la fraction d'eau transpirable du sol (FTSW).

La croissance du pois est déclenchée par le passage au stade 'levée'. La variable d'état visée est l'indice de surface foliaire (LAI). Le calcul est paramétré par les caractéristiques du groupe variétal du pois semé. Il tient

compte du stade de développement de la culture.

De manière plus synoptique, l'ensemble des processus biophysiques sur une parcelle cultivée en pois est représenté dans l'Annexe 2. Le diagramme indique, pour chaque processus, les informations qu'il requiert en entrée et les variables d'état qu'il value par son effet. A cause de la nécessaire linéarité de l'enchaînement des modules, certaines données en entrée sont relatives au pas de calcul précédent (c'est-à-dire au jour précédent). Les variables d'état modifiées le jour j ne peuvent être exploitées avec cette sémantique que par les modules intervenant postérieurement au titre du même pas de temps.

4. Les connaissances externalisées

L'Annexe 3 fournit une description complète des types de données et connaissances externalisées, c'est-à-dire que l'utilisateur peut modifier d'une simulation à l'autre, sans avoir à revenir sur le contenu (internalisé) de la base de connaissance. On notera qu'on a fait usage de la possibilité de déporter partiellement le contenu de certains fichiers dans d'autres, pour en clarifier le contenu et alléger le volume. Par exemple, le fichier des paramètres du système (suffixé par '.par') contient des directives qui lisent trois groupes de paramètres relatifs à trois thèmes différents dans trois fichiers différents.

L'externalisation de données porte sur la météo, les sols, les plantes, la conduite des cultures. Une autre externalisation importante est la structure du système de culture, avec ses éléments blocs-rotation. Chaque bloc-rotation fait l'objet d'une déclaration réduite à la surface des parcelles qui le composent et aux cultures qui y seront pratiquées lors de la première campagne (plus précisément le groupe variétal choisi pour l'espèce visée et l'itinéraire technique envisagé). A partir de ces informations, un mécanisme propre à SILASOL instancie les parcelles (et les placettes, donc les sols et les cultures) du bloc. Pour permettre une expression aussi simple de la structure globale, il a fallu introduire dans la base de connaissance une bibliothèque d'itinéraires techniques prédéfinis que l'interpréteur de la description des blocs-rotation sait reconnaître et mobiliser. Des explications complémentaires sont données dans le chapitre relatif à la conduite du système. .

5. Le modèle de la conduite du système

5.1 Itinéraires techniques

Le modèle de la conduite du système de culture est fondé sur la notion d'itinéraire technique, au sens d'un plan cohérent d'activités culturales mises en œuvre sur une parcelle dans l'intervalle défini par une campagne culturale. Un itinéraire technique est un ensemble d'activités primitives (par exemple un semis ou une récolte), typiquement organisées en une séquence. Une activité primitive est spécifiée par le lieu de mise en œuvre (une parcelle), le type d'effet recherché (l'opération culturale) et les moyens qu'on souhaite affecter à sa réalisation (les ressources). C'est le composant 'opération' qui véhicule le changement d'état visé par le choix de l'activité.

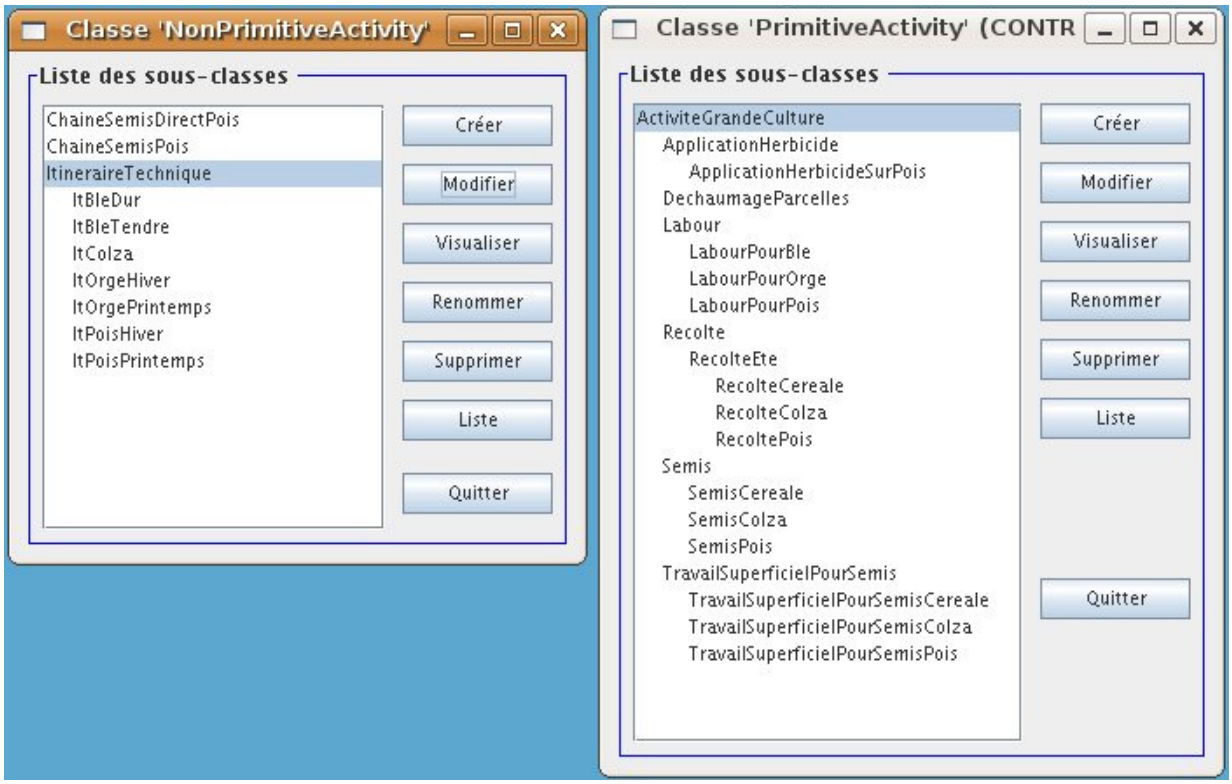
La base SILASOL contient une bibliothèque (extensible) d'itinéraires techniques, différents par les espèces qu'ils sont censés gérer, les séquences d'activités primitives qu'ils organisent, les variantes d'effet que provoquent leurs opérations. On peut aussi différencier des itinéraires par les règles de déclenchement des actions (par exemple, l'itinéraire de conduite d'un blé précoce est différent de celui d'un blé tardif, parce sur l'activité de semis ne se déclenche pas sur le même critère dans les deux cas). Les itinéraires techniques de toutes les parcelles sont assemblés en un seul plan global attaché au système de culture.

La constitution d'un itinéraire technique est codée dans le constructeur de la classe C++ correspondante. Voici, à titre d'exemple, le constructeur de l'itinéraire technique de pois de printemps (en fait, seulement la partie qui établit les activités primitives et leur organisation) :

```
ItPoisPrintemps::ItPoisPrintemps() {
  // …
  Entity* pAct1 = new LabourPourPois();
  AddElement(pAct1);
  Entity* pAct2 = new ChaineSemisPois();
  AddElement(pAct2);
  Entity* pAct3 = new RecoltePois();
  AddElement(pAct3);
};
```

Dans l'état courant de la base, et pour toutes les cultures, seules les opérations de labour, d'installation de la culture et de récolte sont introduites dans les itinéraires techniques (bien que d'autres, telles l'application

d'herbicide ou le déchaumage, soient prévues).



5.2 Conduite concurrente des itinéraires techniques

Une fois choisis dans la bibliothèque, par l'utilisateur du simulateur, les itinéraires sur chaque parcelle (et par conséquent les successions pluriannuelles d'itinéraires dans chaque bloc-rotation), ce n'est que l'aspect « diachronique » de la conduite qui a été spécifié. Il reste alors à développer l'aspect « synchronique », c'est-à-dire le mode de gestion de la concurrence entre les itinéraires qui se déroulent parallèlement sur les parcelles du système. L'intensité de la concurrence dépend de plusieurs facteurs :

- la quantité de parcelles à conduite et leurs superficies
- la vitesse intrinsèque des opérations à exécuter, modulées par la puissance des matériels utilisés
- le système de culture qui engendre des périodes de pointe de travail plus ou moins larges et intenses
- les conditions d'emploi de la main d'œuvre, notamment la longueur de la journée de travail et sa flexibilité
- les propriétés de la main d'œuvre, notamment sa compétence relative aux différentes activités
- la quantité des jours disponibles pour le travail, résultant de facteurs externes aléatoires comme le climat

On peut distinguer deux ensembles de connaissances en gestion des concurrences : celles qui permettent de réduire le nombre de situations de concurrence, et celles qui servent à arbitrer en situation de concurrence.

Pour ne considérer ici que les connaissances internalisées, celles du premier ensemble sont situées dans plusieurs « niches » de la base :

- Une déjà évoquée : la rédaction des itinéraires techniques qui peut spécifier explicitement ou provoquer indirectement des périodes de mise en œuvre plus ou moins décalées, avec plus ou moins de marges de manœuvre.
- La définition d'une opération technique comprend sa condition de faisabilité, qui peut dépendre des conditions météorologiques actuelles ou dans un passé plus ou moins récent, ou encore dans un avenir plus ou moins proche (perçu par prévision). Un fort degré d'exigence de cette condition peut provoquer des situations de concurrence, en réduisant les marges de manœuvre, et inversement.
- Une main d'œuvre moins spécialisée, ou une spécialisation flexible, augmente les possibilités satisfaire la réquisition des activités, et réduit les concurrences en augmentant les marges de manœuvre.
- Comme détaillé dans l'annexe 4, la manière de gérer la fin de la journée de travail est codée dans la définition de l'opération (pour le décalage de l'arrêt du travail pour en terminer une) et par la pose d'une contrainte sur l'opération (pour le non démarrage d'une opération trop longue).
- Le rythme auquel le gestionnaire du système de cultures réexamine ses décisions de pilotage est réglé par les événements de révision de l'état d'ouverture/fermeture d'activités du plan global. On prévoit dans SILASOL un examen complémentaire à la mi-journée pour capter l'éventuelle opportunité de continuer à occuper la main d'œuvre si elle a été libérée dès la matinée, réduisant ainsi les situations de concurrence.
- La spécification des activités primitives inclut l'expression des ressources allouées, et on peut ainsi

privilégier tel ou tel type d'activités, dans tel ou tel contexte. A titre d'exemple, voici la manière d'exprimer, dans le constructeur de l'activité de travail superficiel en vue de semer du pois, qu'on souhaite allouer un ouvrier permanent à temps plein, n'importe lequel, à sa réalisation :

```
TravailSuperficielPourSemisPois::TravailSuperficielPourSemisPois() {
  // ...
  PerformerSpecification* pOuvrierTSSolSpec
    = new PerformerSpecification();
  pOuvrierTSSolSpec->SetIntVarValue(ENTITY_SPEC_REFERENCE_CLASS_ID,
                      OUVRIER_PERMANENT_TEMPS_PLEIN);
  pOuvrierTSSolSpec->SetIntConstValue(SELECTOR_FCT_ID, ANYONE);
  pOuvrierTSSolSpec->SetIntConstValue(SELECTOR_FCT_ARG, 1);

  SetEntityVarValue(PERFORMER_SPEC_ATTRIBUTE, pOuvrierTSSolSpec);
};
```

Pour arbitrer en situation de concurrence, le degré de priorité des opérations est exploité par le moteur de simulation pour ordonner la réalisation pratique de plusieurs activités que l'interprétation du plan a jugé opportun de mettre en œuvre, mais que la limitation des ressources ne permet pas de réaliser conjointement. Les opérations ainsi retardées redeviendront candidates plus tard. Les priorités des opérations actuellement dans la base sont les suivantes, par ordre croissant de priorité :

```
RecolterColza : 97
RecolterPois : 95
RecolterCereale : 91
PasserHerbicide : 80
SemerPois : 75
PasPasserCoverCrop : 75
Labourer : 71
Decompacter : 70
SemerCereale : 61
SemerColza : 61
```

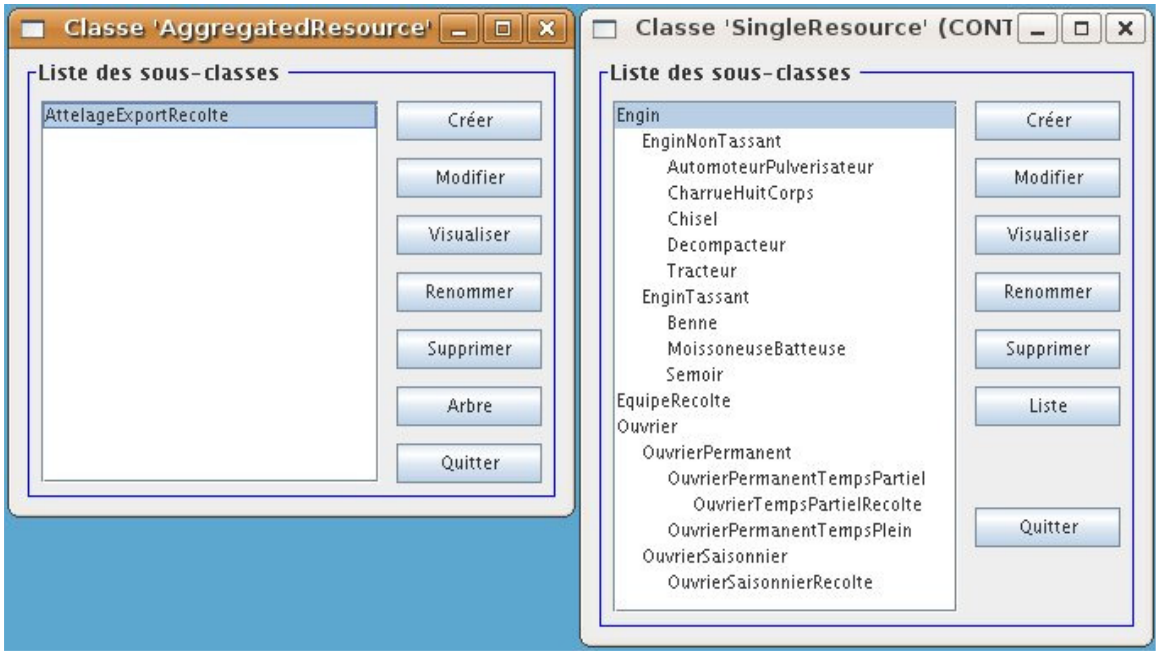
En dernier ressort, on résout les concurrences entre activités candidates en 'surchargeant' de manière appropriée la méthode 'BestActivitySetSelector' de l'instance de la classe Manager, un des composant du système de production (Figure 1). Cette méthode désigne, parmi plusieurs jeu d'activités candidates à l'exécution, donc munis de ressources disponibles, celui qui est globalement préféré. Dans sa version actuelle, cette méthode choisi le jeu qui possède l'activité dont l'opération est de plus grande priorité parmi toutes les opérations dans tous les jeux. Autrement dit, on a reporté au niveau de la gestion tactique l'usage d'un critère que le moteur de simulation exploite par défaut au niveau opérationnel.

(à compléter)

5.3 Constitution et gestion du pool des ressources

Un autre aspect de la conduite du système est la constitution de l'ensemble des ressources potentiellement disponibles sur l'exploitation, leurs propriétés et les relations d'agrégation qui les lient éventuellement (ressources dites 'agrégées', telles que l'attelage export-récolte). Ces ressources sont le matériel et la main d'œuvre. Les classes développées sont dans la figure ci-dessous.

Sur la ressource 'main d'œuvre', plus de détails sont donnés dans l'annexe 4, sur la classification adoptée, sur la spécification du programme de travail de chaque unité de main d'œuvre, sur les contraintes posées sur son utilisation (compétences, partageabilité, et sur la manière de gérer la poursuite ou l'entame d'une activité en fin de journée de de travail.



(à compléter)

7. Les modules rapportés : tassement du sol et sensibilité au gel

Les modules de calcul du tassement du sol et de sensibilité du pois au gel n'étaient pas intégrés dans le modèle AFISOL. Plus précisément, le tassement du sol était un paramètre d'entrée, non calculé dynamiquement après chaque opération. Et le module « gel » faisait l'objet de quelques formules dans le tableur AFISOL, sans que les valeurs calculées soient reprises dans le modèle de croissance du pois (en principe dans sa variable epsilon_b).

7.1 Le tassement du sol

Pour l'intégration du modèle de tassement, on a extrait du logiciel SISOL le cœur de calcul, c'est à-dire le code non dédié à l'interface-utilisateur. A ce code C++, on a donné la forme d'une fonction qui renvoie le « pourcentage de zone tassée » en l'instant de son invocation. Et on a prévu d'invoquer une des méthodes de la classe Operation, prédéfinies dans CONTROL DIESE, la « fonction de transition d'état » des différentes opérations susceptibles d'entraîner un tassement préjudiciable au pois.

La transcription du code de SISOL, puis le test de cette transcription hors du contexte SILASOL, et enfin l'analyse des résultats ont fait émerger un questionnement sur le contenu du module (cf.Annexe 5) et la validité des valeurs calculées. Cela a conduit au débrayage de cette fonction dans SILASOL, et son remplacement par l'affectation d'une valeur arbitraire, ayant donc le statut de variable d'entrée, comme dans AFISOL.

Vu l'importancedu facteur 'tassement' dans le comportement de la culture du pois, la réintroduction du module SISOL est sans doute nécessaire.

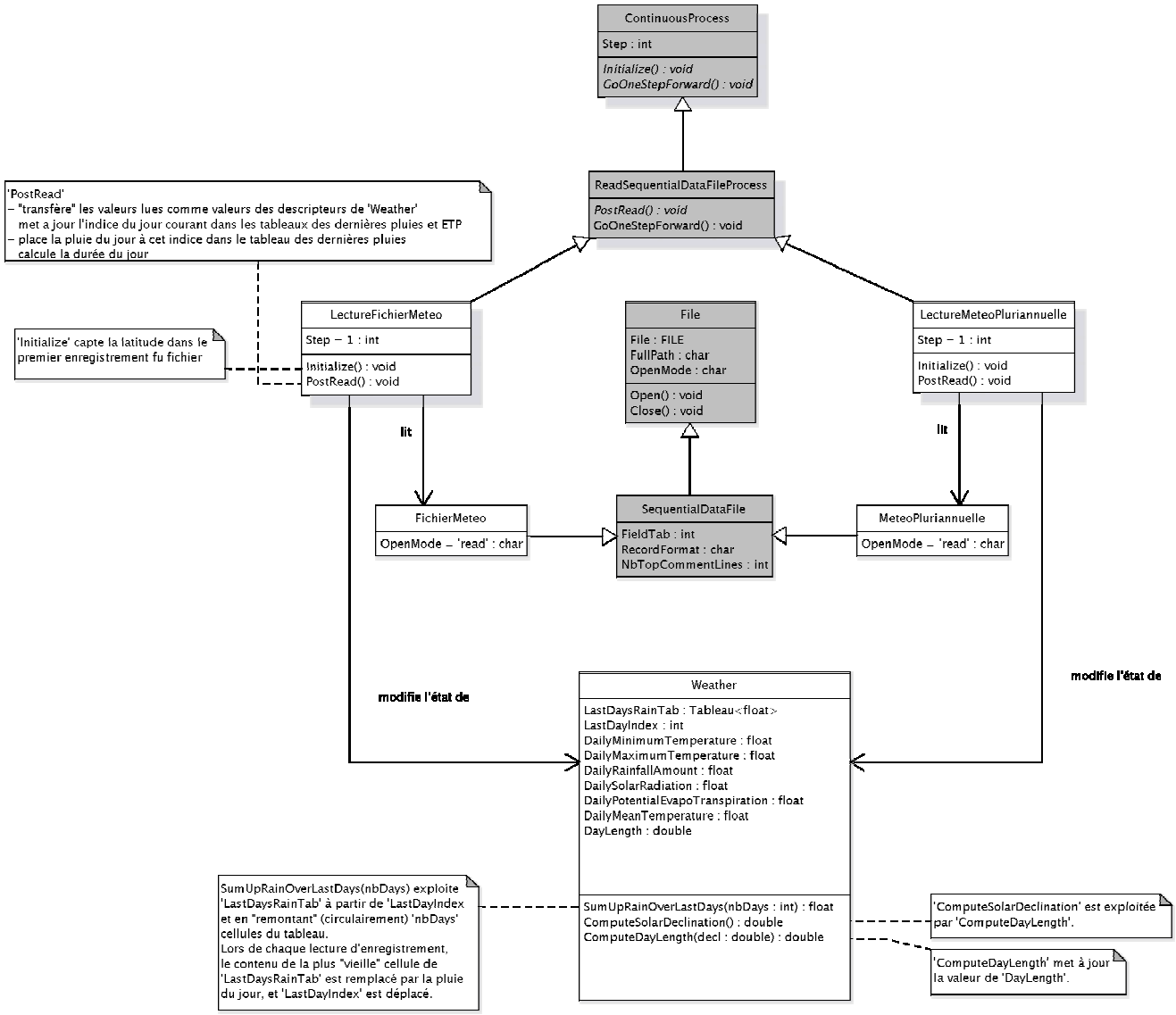
7.2 La sensibilité du pois au gel

Dans SILASOL, le calcul de la valeur epsilon_b du modèle de croissance fait intervenir de manière distincte un facteur « thermique » et un facteur « gel », alors que seul le premier des deux est présent dans AFISOL. Un processus, lancé au moment du semis, calcule quotidiennement le facteur gel dans une plage [0. 1.] (d'aucune réduction à l'annulation totale de l'augmentation du poids de MS pour le jour courant).

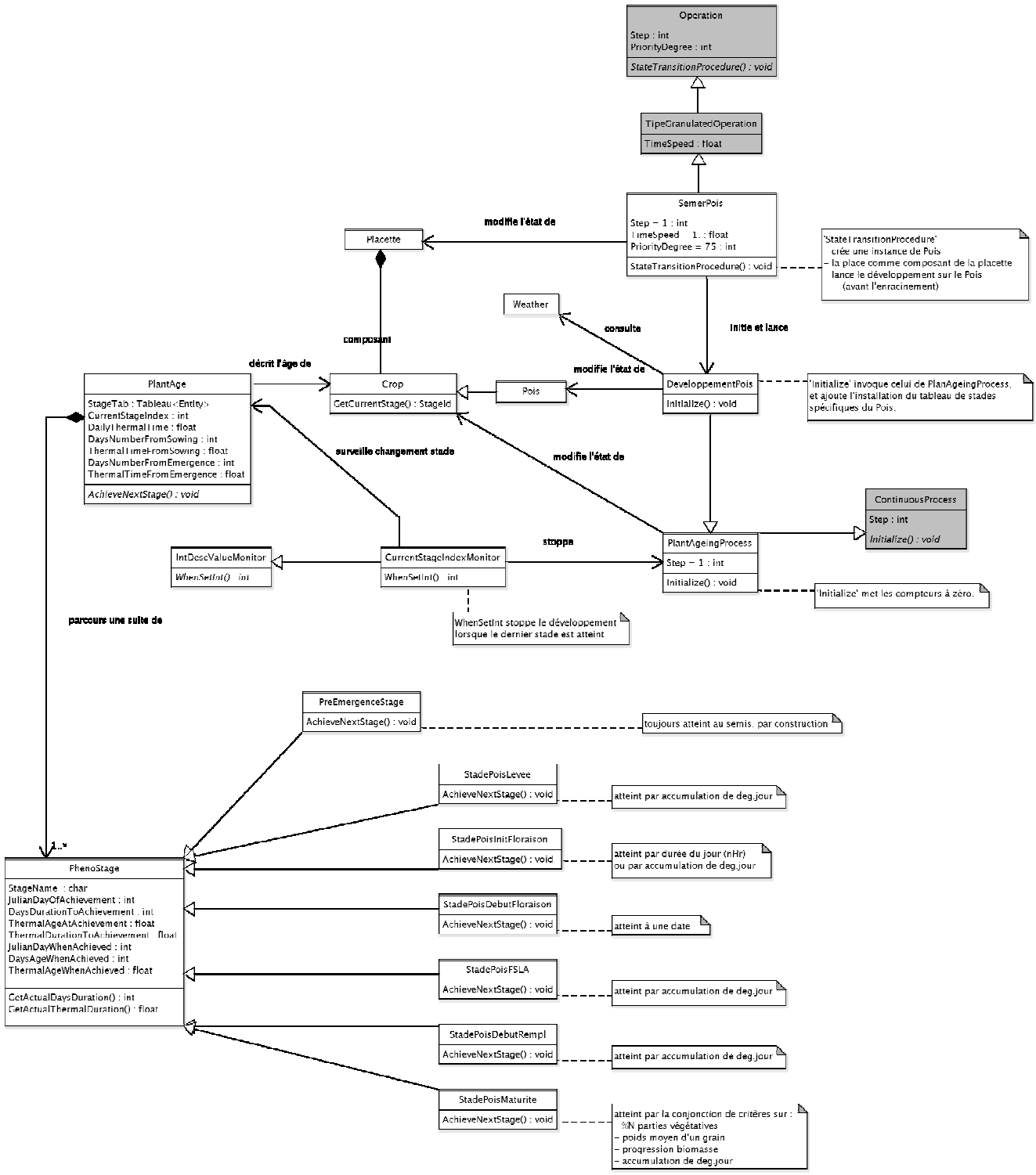
L'Annexe 6 est le document qui a servi de base à l'analyse. Cette analyse, et sa traduction dans la base SILASOL, n'a pas encore été validée. Le calcul du facteur « gel » est donc débrayé (avec la valeur 0.).

Annexe 1 : représentation UML des processus biophysiques de SILASOL

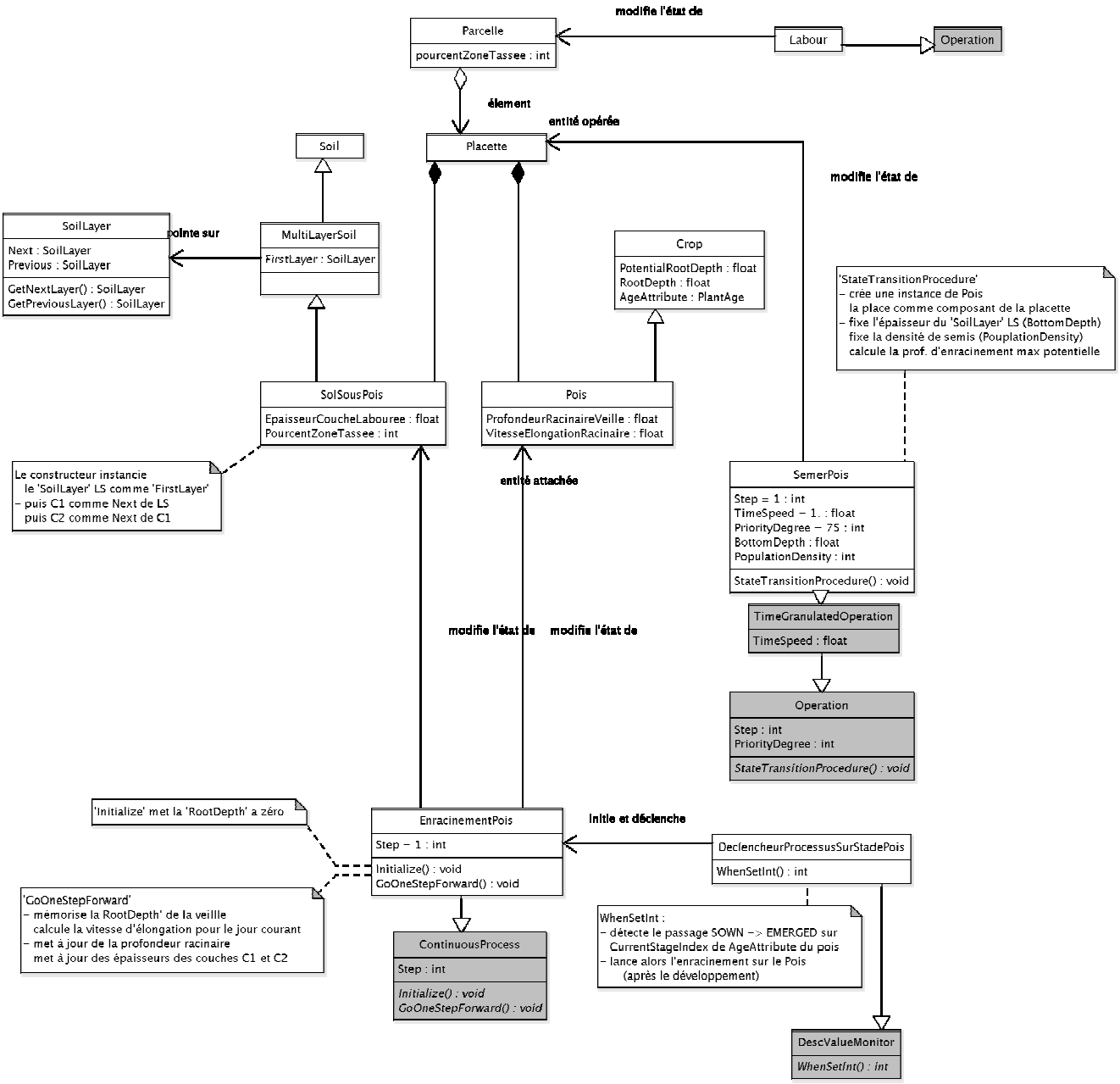
L'évolution météorologique



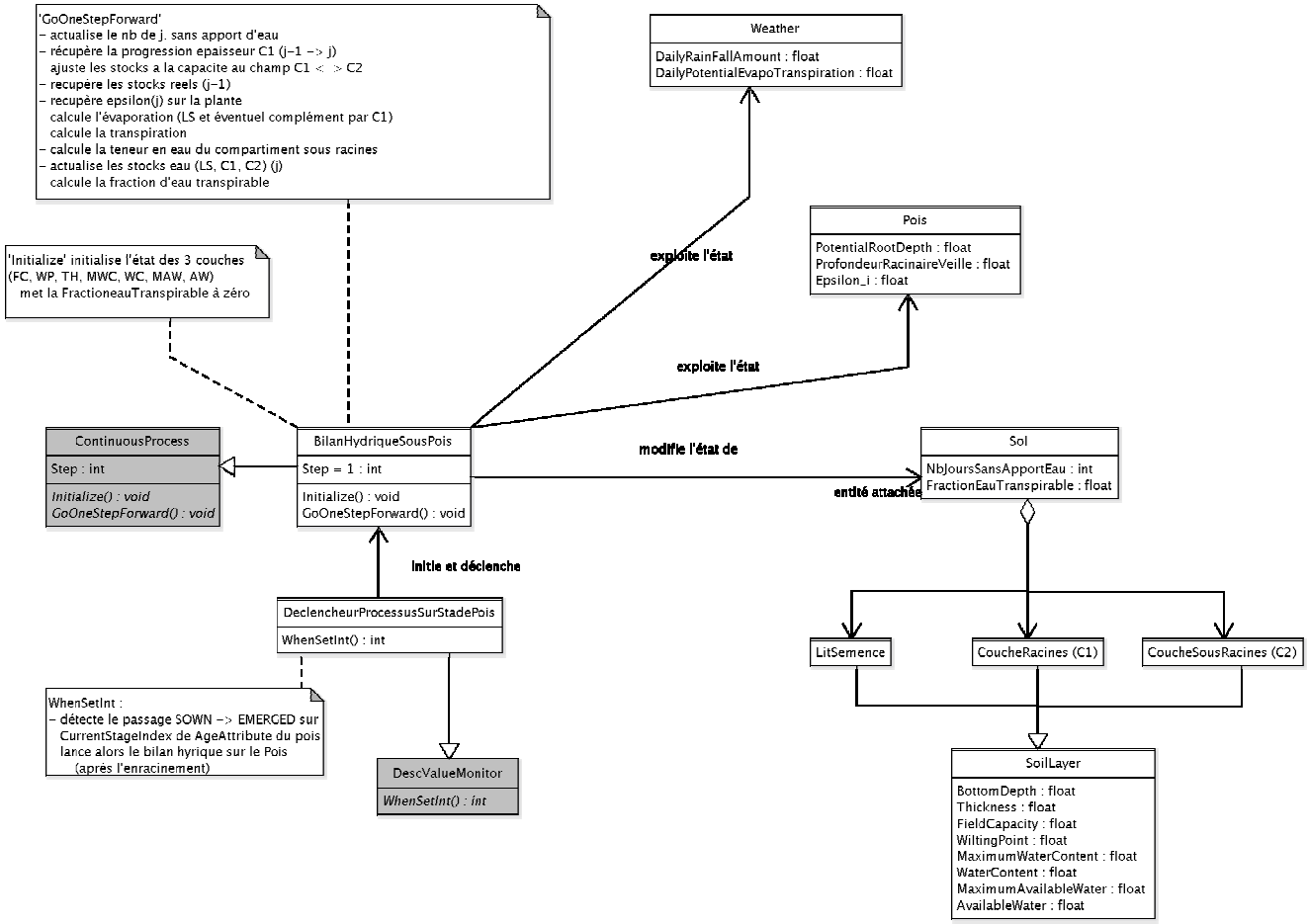
L'évolution de l'âge des plantes



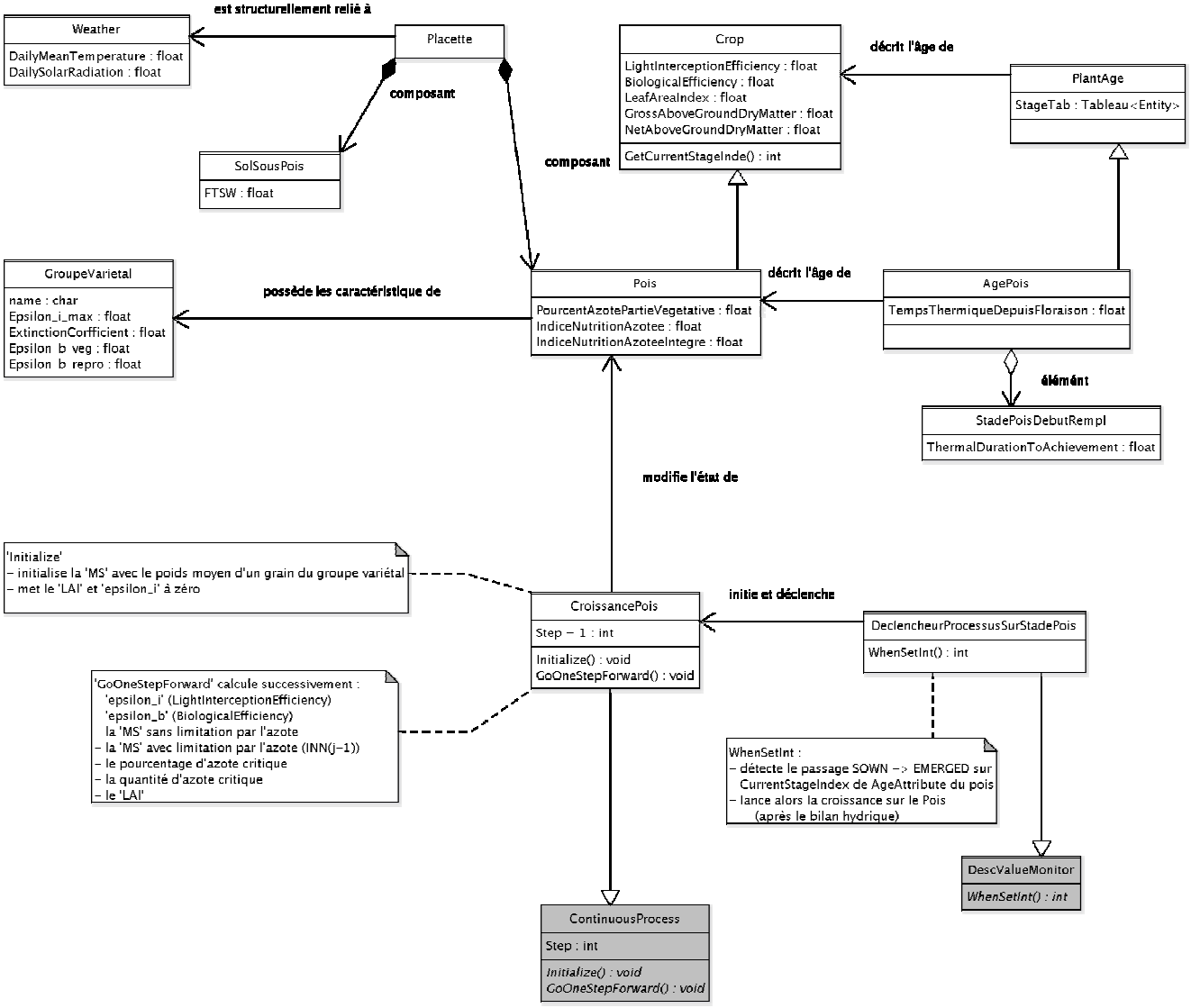
La progression de l'enracinement



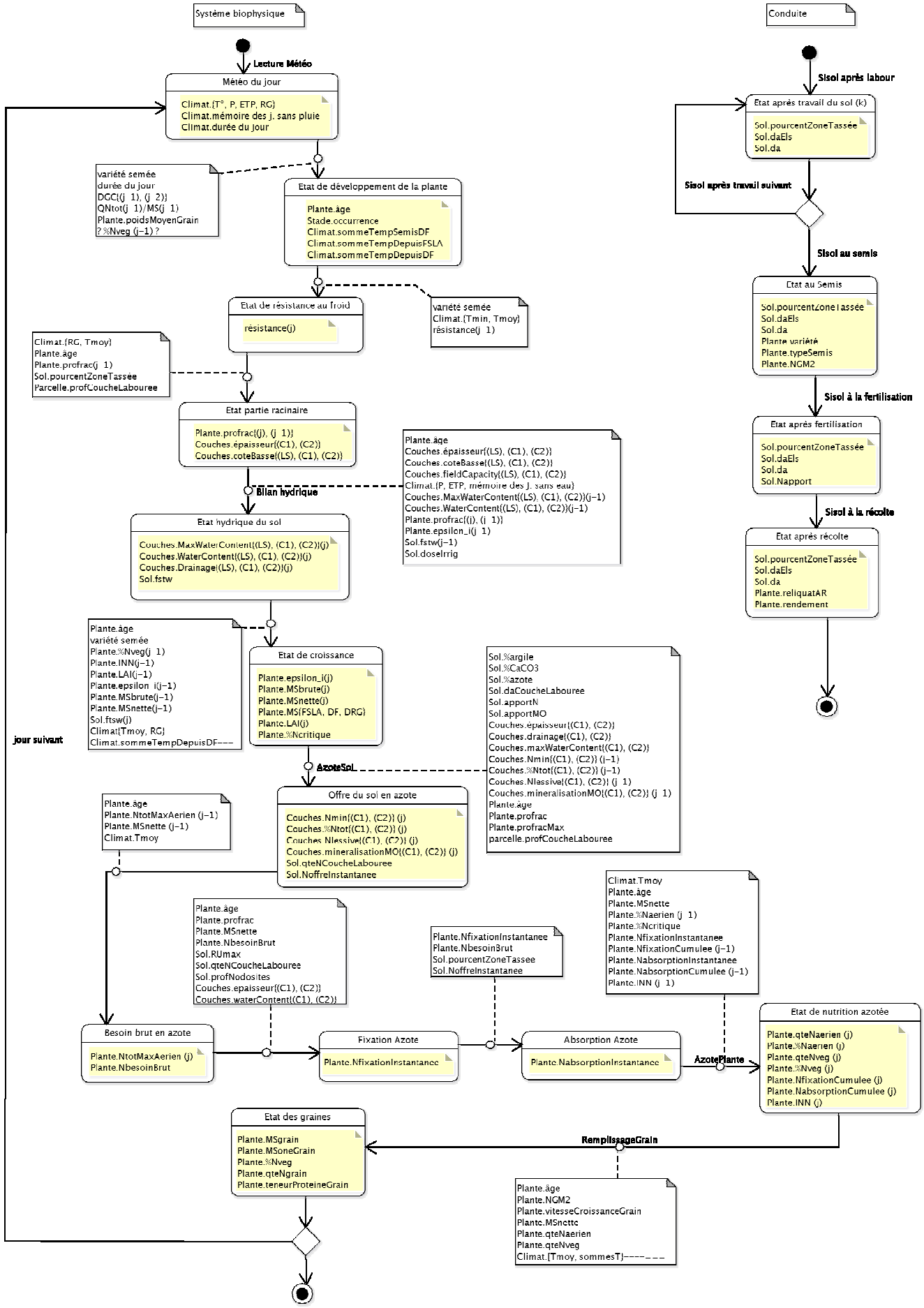
La satisfaction du besoin en eau



La croissance



Annexe 2 : diagramme d'état d'une parcelle cultivée en pois



Annexe 3 : connaissances externalisées dans les fichiers en entrée

```
Vue générale sur les fichiers d'entrée, arguments de la commande 'main' :
-------------------------------------------------------------------------

test.par
             <- parcelles_sols.inc           : pour chaque parcelle
                                                        . granulométrie, %N, ...
             <- conduite_options.inc         :
             <- plantes_afisol.inc           : paramètres du modèle AFISOL (Pois)
             :
test.sim
             :
test.osp
             :
test.str
             <- sols_typesTextures.inc       : pour chaque texture (S, SL, SA, etc.)
                                                     . intervalles pour clay, silt, sand
                                                     . FC et WP gravimétriques
             <- plantes_phasicValueSets.inc  :
             <- plantes_groupesVarietaux.inc : pour chaque groupe variétal
             <- conduite_ressources.inc      :
             <- conduite_blocs.inc           :
             :
test.dir
             :


Environnement climatique :
--------------------------
* Deux types de fichiers de valeurs journalières peuvent être utilisés,
différenciés par le jeu de variables météo.

1) les fichier dits 'meteo_*' ou 'met_*' ne présentent que les champs communs :
DATE          ETP   Tmoy   Rg           P      T°max

2) les fichier dits "climat_*" ont un champ additionnel :
DATE          ETP   Tmoy   Rg           P      T°max  T°min

Selon le type de fichier utilisé en entrée, on exploitera l'un ou l'autre
des deux classes de processus 'lectureFichierMeteo' ou 'lectureFichierClimat'
(voir ci-après). Le rôle de ces processus est de valuer quotidiennement les
descripteurs (pluie, RG, etc...) d'une instance de la classe FichierMeteo
ou de la classe FichierClimat, selon le cas.

A noter : ces classes de fichiers héritent de la classe SequentialDataFile et
non de la classe BufferizedDataFile (toutes les deux prédéfinies dans cdiese).
Deux conséquences :
- la gestion de la mémoire des dernières pluies est programmée 'ad hoc',
alors que la classe BufferizedDataFile aurait fourni des services prédéfinis.
- l'agriculteur est supposé ne pas avoir de capacité de prédiction sur la
météo (ce que la classe BufferizedDataFile aurait permis de gérer).

Quel que soit le type, le fichier doit débuter par 3 lignes de commentaire
libre, mais qui présentent normalement le contenu du fichier.
La première ligne doit contenir la valeur de LATITUDE en degrés, dans le
format  suivant : <texte libre> LAT 50.0 <texte libre>

* La déclaration du fichier exploité est faite dans le fichier '*.str',
lors de la spécification du processus de lecture du fichier :
REPERTOIRE_FICHIER "<repertoire contenant le fichier>"
NOM_SIMPLE_FICHIER "<nom simple du fichier>"


Les données "sols" :
--------------------
Chaque parcelle est caractérisée par un jeu de valeurs portant sur son sol
(granumométrie, pourcentage d'azote, quantité de matière organique apportée).

Ces valeurs sont déclarées dans le fichier de paramètres 'parcelles_sols.inc'
(inclus par le fichier-racine des paramètres : test.par) et attribuées au sol
dans le moniteur sur la superficie de la parcelle.
```

Lors de l'attribution des valeurs de granulométrie, est automatiquement déterminé l'unique type du sol (e.g. typeTexture_SL), en fonction de la caractérisation de ces types faite dans le fichier 'sols_typesTextures.inc'.

La caractérisation des types de textures est complétée par les valeurs des pourcentages d'humidité pondérales à la CC et au PF4.2.

Pour tous les sols : une valeur unique pour les variables pourcentageCACO3 et densiteApparenteCoucheLabouree, chacune fixée dans le fichier *.par.

Les données "plantes" :
---
* Les paramètres spécifiques au Pois sont dans 'plantes_afisol.inc' (inclus par le fichier-racine des paramètres : test.par).

* De manière générale, les paramètres "plante" des cultures sont dans les spécifications des groupes variétaux (fichier plantes_groupesVarietaux.inc, inclus par le fichier-racine de la structure du système : test.str).
Chaque culture est caractérisée par un groupe variétal lors de la désignation de son itinéraire de conduite (fichier conduite_blocs.inc).

La caractérisation d'un groupe variétal peut comprendre les valeurs que prennent certaines variables à différents stades phénologiques.
Un jeu de valeurs propre à un stade est un "phasicValueSet".
Un groupe variétal est doté de zéro, un ou plusieurs phasicValueSet's, chacun se rapportant à un stade particulier.

Les phasicValueSet's (à l'usage de l'ensemble des groupes variétaux d'une espèce) sont déclarés dans le fichier 'plantes_phasicValueSets.inc'.
Dans la version courante, les seuls phasicValueSet's déclarés sont des instances de la classe PhasicCropCoefficientValue, dont le jeu de variables ne comporte que le coefficient cultural pour l'évapotranspiration.
Ainsi, chaque phasicValueSet est caractérisé par la valeur du stade en jeu et par la valeur du coefficient cultural pour ce stade.

Les données "conduite" :
---
* Les ressources sont déclarées dans le fichier 'conduite_ressources.inc'.
Ce sont les ressources mobilisables dans les activités, c'est-à-dire que certaines d'entre elles peuvent ne pas être mobilisées par telle ou telle conduite particulière, ou ne pas être mobilisées du tout.
La déclaration des ressources requises par les activités est internalisée :
- dans les constructeurs des opérations attachées aux activités, pour les ressources propres d'opérations
- dans les constructeurs des activités, pour les 'performers'

* Les contraintes sur les ressources sont déclarées dans le même fichier.
Les contraintes effectivement mises en jeu sont attachées au système opérant dans le fichier '*.str'.

* Les itinéraires techniques sont internalisés dans une bibliothèque de classes d'activités non primitives (sous-classes de ItineraireTechnique).
Leurs constructeurs instancient des classes d'activités (primitives ou non), et les relient par des opérateurs (typiquement de séquencement : Before).
Au-delà de cette structuration temporelle, les propriétés des activités et des itinéraires(et des opérations sous-jacentes) sont parfois internalisées (e.g. le délai entre labour et semis de pois), et parfois externalisées (e.g. modalités de réalisation des opérations ou des activités dans 'conduite_options.inc': vitesse, ressources requises, dates, etc.).

La description du système de production :

Elle est faite dans le fichier '*.str'.

* En préambule, le système de production (exploitationAgricole) est déclaré vide, puis sera doté de ses composants après leur construction dans un ordre quelconque.
Il est en même temps désigné comme l'entité simulée.

* Le système biophysique est doté de son composant 'environnement', instance de la classe Weather. Puis il est attaché au composant 'controlledSystem' du système de production.

Le composant 'systemeDeCultures', avec sa structure en blocs-rotations et la déclaration des conduites pour chaque parcelle, sera attaché après la déclaration du manager et de sa stratégie encore vide.
Une fois la stratégie ainsi construite, le plan est rendu prêt à être exécuté, par l'attribution d'une valeur à readyToRun, doté du moniteur programmé à cet effet dans cdiese.

* L'unique composant du système opérant est un pool multiple de ressources. Celui-ci a deux éléments (pools simple de ressources) : le personnel et le matériel, déclarés dans 'conduite_ressources.inc'.
Un sous-ensemble des contraintes déclarées dans 'conduite_ressources.inc' est ici mis en jeu.

* Le manager est doté d'une stratégie incluant un plan d'activités encore vide. On déclare en outre sa capacité de mémoire utile de la météo (en nb de jours).

* Le système de production est doté de ses composants maintenant instanciés.

* La déclaration du parcellaire (blocs-rotations et parcelles) est faite dans le fichier 'conduite_blocs.inc'.
On y instancie chaque bloc-rotation, qu'on décrit par tois caractéristiques :
  . la liste des superficies des N parcelles composant le bloc
  . la séquence des noms des N itinéraires techniques formant la rotation sur ce bloc
  . la liste des noms des N types variétaux cultivés par chacun de ces itinéraires

Le passage à la valeur 1 du descripteur 'readyToInstall' du systemeDeCultures déclenche un moniteur qui examine tour à tour les blocs-rotation et provoque sur chacun d'eux les effets de bord suivants :

- avec la liste des superficies en ha (<u.d> e.g. 10.5) des parcelles ... :
  . le moniteur sur 'readyToInstall' instancie autant de parcelles qu'il y a de valeurs de superficie
  . on structure chaque parcelle instanciée en ceil(u) (ici 11) placettes de 1 ha (la dernière fait 1 ha ou bien 0.d ha -ici 0.5-).
    Seule la première placette sera l'objet des processus biophysiques.
  . un autre moniteur sur la superficie dote chaque placette de deux sols, un multi-couche et un mono-couche.
    Ces deux sols seront alternativement et exclusivement exploités par les processus du Pois d'une part, et par les processus des autres cultures et du sol nu. d'autre part.
    La transmission réciproque de valeurs de descripteurs lors des "relais" est codée.
  . ce même moniteur lance les processus de bilan hydrique simple et de minéralisation de l'azote sur le sol mono-couche.
    A noter : ceci implique que l'état initial du système doit en principe correspondre à une date où toutes les parcelles sont en sol nu. S'il est impossible de désigner cette date, dans telle ou telle circonstance (par exemple si le système de culture comportait des cultures d'été), la ou les parcelles portant une culture à la meilleure date choisie devront être quand même simulées en sol nu. La dynamique simulée du système est alors erronée jusqu'à la date à laquelle la dernière des cultures en question aurait été récoltée.:

- avec la séquence des noms des itinéraires techniques attachés au bloc ... :
  . le moniteur sur 'readyToInstall' traite successivement chaque nom et ...
    ... instancie la classe d'itinéraire portant ce nom,
    ... le relie aux autres par un opérateur de séquencement (before) pour constituer la rotation
    ... encapsule la rotation dans un opérateur de répétition (iterate) pour constituer la gestion à long terme
    ... attache cette gestion à long terme à la parcelle dont la superficie est du même rang que le nom d'itinéraire considéré (en désignant la parcelle comme "l'objet opéré" de chaque activité primitive)

      Lors de l'examen de chaque nom d'itinéraire, le moniteur fait débuter la séquence d'itinéraires par cet itinéraire, de telle sorte que, sur le bloc, tous les itinéraires soient opérés chaque année de simulation.

    ... ajoute enfin cette gestion à long terme au plan global d'activités attaché au manager

      Pour illustration, considérons le bloc suivant :

```
+ I blocRotation B1
       superficiesDesParcelles <<      10.0           16.0       25.0;
       conduitesDesParcelles << "itPoisPrintemps" "itBleHiver" "itColza";
       groupesVarietauxDesCultures <<
                            "poisPrintemps_PH2"  "bleTendre_type1" "colza_type1";
;
       Alors, les "gestions à long terme" sont de la forme :

              iterate(                          // pour la parcelle de superficie 10.0
                           before(      itPoisPrintemps,itBleHiver,itColza))
              iterate(                          // pour la parcelle de superficie 16.0
                           before(      itBleHiver,itColza,itPoisPrintemps))
              iterate(                          // pour la parcelle de superficie 25.0
                           before(      itColza,itPoisPrintemps,itBleHiver))

- avec la séquence des noms des types variétaux cultivés ... :
  . le moniteur sur 'readyToInstall' attache un type variétal à chaque
    itinéraire, qui sera exploité par l'opération sous-jacente de semis.

* Un "itinéraire technique" peut être une simple séquence (semis ... récolte),
ou bien une disjonction de deux séquences (""itineraireBleDurOuBleTendre"),
ou un autre activité non primitive (optionalité, ...).

* La construction des itinéraires de la bibliothèque peut exploiter des morceaux
d'itinéraires communs, qui font alors l'objet d'une classe réutilisable.


La spécification et la programmation des processus :
----------------------------------------------------
* Les seuls processus explicitement gérés dans le fichier '*.str' sont la
lecture du fichier météo et le processus d'interprétation du plan d'activités.

* Les autres processus (biophysiques) sont lancés ... :
  . ... dans le moniteur sur la superficie des parcelles (voir ci-dessus)
  . ... et lors du semis des cultures : c'est le processus de développement.
Pour le Pois, ce processus de développement gère la dynamique des autres
processus, lequels sont déclenchés au semis, à la levée ou encore à FLSA.
Les opérations de récolte fonc passer le stade à 'destroyed', ce qui provoque
l'arrêt des processus sur la plante, et un relais du sol mono-couche au sol
multi-couches.

A noter : le processus LancerFournitureAzoteSol, et l'événement correspondant
LancementFournitureAzoteSol, ne sont plus exploités. Leur rôle est désormais
joué par le moniteur sur la superficie de la parcelle.
```

Annexe 4 : le modèle de l'utilisation de la main d'œuvre

L'organisation du travail (agricole) est l'ensemble des dispositions prises (de manière anticipatoire, planifiée ou réactive) par le gestionnaire de l'exploitation, pour affecter au mieux la main d'œuvre à la réalisation des opérations, compte tenu de contraintes et de préférences (i) sur la disponibilité et la compétence des éléments de main d'œuvre, (ii) sur la disponibilité d'autres ressources qui doivent être co-engagées avec les éléments de main d'œuvre dans la réalisation des opérations.

Classes de ressources 'main d'œuvre' :

Deux classes principales liées au STATUT :
- les ouvriers permanents : ils sont automatiquement mobilisables dès leur entrée dans le système (instanciation). Le programme de travail est la répétition de l'organisation hebdomadaire, tout au long de l'année.
- les ouvriers saisonniers : ils ne sont pas automatiquement mobilisables dès leur entrée dans le système, mais seulement par l'occurrence d'un événement particulier (typiquement dédié à cette mobilisation). Le programme de travail hebdomadaire est répété dans les périodes ainsi générées.

Des sous-classes liées au PROGRAMME ANNUEL DE TRAVAIL :
- les ouvriers à temps plein : des événements dédiés peuvent générer des périodes d'immobilisation (vacances ou autres absences). Il n'y a pas de tels événements dans les jeux de données actuellement exploités.
- les ouvriers à temps partiel : des événements dédiés génèrent des périodes d'immobilisation (vacances ou autres absences). Il s'agit actuellement d'une série de 4 paramètres (conduite_options.inc) spécifiant le début (jour et mois) et la fin de la période de vacances (scolaires) d'été.

Des sous-classes liées aux COMPÉTENCES :
- on a identifié la classe des ouvriers temps partiel affectés aux récoltes. Il conviendrait de poser une contrainte de type ActivityInconsistancyCondition sur toute activité, imposant un tel ouvrier d'être co-engagé, dans cette activité, avec une opération de récolte ; cela n'a pas été programmé. Dans l'état actuel de développement, le respect de la compétence est de la responsabilité du modélisateur, lorsqu'il rédige la PerformerSpecification des différentes activités. On peut en effet vérifier que ce n'est que dans la classe SpecificationOperateurRecolte qu'est mobilisée une instance des classes OuvrierSaisonnierRecolte et OuvrierTempsPartielRecolte.
- on pourrait, presque de même, doter les ouvriers de contraintes sur l'utilisation des différents matériels, ou sur l'intervention sur les différentes cultures ou parcelles, etc.

Gestion des alternances des périodes de mobilisation et d'immobilisation :

La mobilisation d'un ouvrier est nécessairement suivie d'une immobilisation, sauf si cette dernière devait intervenir après la fin de la période simulée. Une immobilisation est généralement suivie d'une mobilisation (avec la même restriction que ci-dessus), tant que la ressource en jeu est dans le système et tant qu'aucune situation particulière ne justifie la non-(re)mobilisation.

Pour reproduire cette alternance, on exploite un couple d'événements prédéfinis dans CONTROL DIESE : MobilizationEvent et ImmobilizationEvent, dont on a créé des spécialisations dans SILASOL. En effet, ces deux événements sont dotés de méthodes dédiées à la génération de l'événement 'dual' (le moteur de CONTROL DIESE prenant en charge la mécanique d'insertion dans l''agenda d'événements). La connaissance sur l'enchaînement des mobilisations/immoblisations des ouvriers est donc codée dans ces deux méthodes. Plus précisément, à chaque classe d'ouvriers correspond un couple d'événements. Par exemple, à la classe OuvrierSaisonnierRecolte correspond le couple 'MobilisationSaisonnierRecolte / ImmobilisationSaisonnierRecolte'.

Pour suivre cet exemple, la méthode GenerateNextEvent de la mobilisation date l'immobilisation à la fin de la journée de travail qui vient de démarrer, en tenant compte de la durée du jour de travail, spécifiée par un paramètre dont le premier identifiant est "dureeJourTravail". Son second identifiant dépend de la période en cours ("semaineNormale", "semainePointe", "samediPointe").

La méthode GenerateNextEvent de l'immobilisation date la mobilisation. C'est généralement le lendemain, à l'heure fixée pour l'embauche, supposée précéder très immédiatement le moment de la révision quotidienne du plan par le

gestionnaire de l'exploitation. Elle est donc spécifiée par le paramètre '"revisionDuPlan" "heureMatin"'. La "règle du lendemain" n'est pas respectée lorsque ce jour n'est pas ouvré pour l'ouvrier considéré ; la mobilisation est alors repoussée au matin du prochain jour ouvré pour l'ouvrier et la période considérée (typiquement le lundi).

Initialisation de l'alternance mobilisation/immoblisation :

Pour les ouvriers permanents, la première mobilisation est programmée dans le constructeur de la classe (l'événement correspondant est donc placé dans l'agenda par l'instanciation). Pour les ouvriers à temps plein, la date est la date d'instanciation. Pour ceux à temps partiel, la date est le premier jour ouvré suivant l'instanciation.

Pour les ouvriers saisonniers, seul le cas des "récolteurs" a été considéré.
* La première mobilisation intervient au début de la période des récoltes. Ce moment est défini comme la date à laquelle la première atteinte du stade 'maturité' est observée pour une culture de printemps (récoltée en été). Ce moment est détecté par un moniteur placé sur le descripteur CurrentStageIndex de la classe d'entités SpringPlantAge (dont une instance est la valeur du descripteur AgeAttribute de toute culture de printemps). A ce moment, le descripteur IndicateurRecolteEnCours du Manager devient 'vrai'. Un autre moniteur placé sur ce descripteur déclenche à son tour la mobilisation des ouvriers déjà instanciés.
* L'IndicateurRecolteEnCours devient 'false' lorsque la dernière récolte d'été est fermée (détection par un moniteur sur le descripteur Situation de la classe RecolteEte). Le même moniteur qui a déclenché la première mobilisation déprogramme alors la prochaine mobilisation des ouvriers saisonniers.

Nota : la valeur du descripteur IndicateurRecolteEnCours du Manager est par ailleurs exploitée pour :
- calculer la date de la prochaine mobilisation, parce que l'ensemble des jours ouvrés depend de la période en cours,
- connaître la longueur de la journée de travail, pour calculer les dates d'immobilisation en fin de journée,
- établir l'heure de la mi-journée de travail, pour programmer à ce moment un examen complémentaire du plan (voir plus loin).

Contraintes d’utilisation de la main d’œuvre :

Une contrainte naturelle est posée sur la main d’œuvre : aucune unité de main d’œuvre ne peut réaliser au même moment deux opérations différentes.
Ceci est exprimé en ajoutant une contrainte de partageabilité au système Opérant.

La vitesse et la durée des opérations :

Ces deux données servent, classiquement, à étaler de manière réaliste l'effet des opérations et à établir la période d'immobilisation des ressources mises en jeu. Pour ce qui concerne l'organisation du travail, elles servent aussi à gérer la fin de la journée (voir la section suivante),

La longueur du 'pas' de l'opération est (ontologiquement) défini de telle sorte que la progression de l'opération puisse être remise en question au terme de chaque pas (et jamais pendant), pour être soit poursuivie soit interrompue. Cette longueur est fixée à 2 heures dans SILASOL : on ne sort pas du matériel pour moins de 2h, et on ne peut pas "ne rien faire" pendant plus de 2h (dans le cas limite d'un pas "occupé" seulement quelques minutes).

La vitesse de l'opération est (ontologiquement) l'augmentation du degré de progression en un 'pas'. Dans SILASOL, les options suivantes sont prises :
- la vitesse est exprimée en nombre de placettes de 1 ha (appartenant à la parcelle qui est l'objet de l'activité). C'est dire que toutes les opérations sont des spécialisations de UnitGranulatedOperation de CONTROL DIESE.
- le nombre d'hectares naturellement exprimé est celui qu'on peut traiter "en une journée de travail normale". On décide d'appliquer cette convention lors de la spécification de la vitesse (descripteur Speed) des opérations, bien qu'elle ne corresponde pas à la définition ontologique. La compatibilité avec la convention ontologique sur laquelle est programmé le moteur de simulation est assurée par la méthode prédéfinie SetUserUnitSpeed, dont le corps surchargé (invoqué à chaque besoin de la vitesse) applique la formule suivante :

speed = 'vitesse nominale' * 'pas' / 'durée jour travail normal'

La valeur de 'durée jour travail normal est celle du paramètre évoqué plus haut "dureeJourTravail" "semaineNormale".

Par exemple, si ce paramètre est valué à 8h, une vitesse exprimée de 50 sera recalculée en 12.5, ramené à la valeur entière 13 (ha par 'pas' de 2 heures).

L'ensemble de ces options sont installées dans la classe d'opérations OperationParHectare, dont toutes les autres (recolter, etc.) sont des spécialisations.

La durée d'une opération est le rapport du nombre d'hectares à traiter sur la vitesse de l'opération. Par exemple, si la vitesse est 13, l'opération sur une parcelle de 25 ha aura une durée de 2 (c'est à dire 2 'pas', autrement dit 4 heures). On parle de durée nominale, parce que la durée effective peut être augmentée par des pauses dans l'exécution. Pour rendre lee calcul de la durée de l' opération en jeu commun à toutes les activités, on a créé la classe ActivitéGrandeCulture, dont héritent toutes les autres.

Gestion de la fin de la journée de travail :

Des mécanismes particuliers sont installés dans SILASOL pour gérer la fin d'une journée de travail, avec les règles suivantes :
(i) alors qu'on démarre une opération, si sa durée nominale est telle qu'elle ne sera que "presque" terminée à la fin normale de la journée, on retarde le moment d'immobilisation des ouvriers mis en jeu ;
(ii) alors qu'on est presque en fin de journée et qu'on s'apprête à démarrer une opération, s'il reste trop peu de temps pour pouvoir la terminer, alors on ne la démarre pas.

L'application de la règle (i) est faite dans le moniteur sur le degré de progression de l'opération (descripteur hérité AchievementDegree), lorsqu'on passe d'un degré nul (au début) au premier degré non nul. La durée nominale est calculée en connaissance de la parcelle opérée, et donc du nombre de placettes à traiter.

La règle (ii) est appliquée en posant une contrainte de consistance sur toute activité de la classe ActivitéGrandeCulture : PasDeOperationEntameeTropTard. Cette contrainte n'est mise en jeu que si l'operation n'est pas entamee, s'il est trop tard dans la journee et si on ne peut pas la terminer avant ce terme (c'est codé dans le corps de la méthode prédéfinie HoldingCondition). Cette contrainte interdit alors la mise en œuvre d'une quelconque opération, par le jeu de l'OccurrenceCrossDomain ConditionDeOperationInterdite.

Examen complémentaire du plan en milieu de journée

Le rythme standard de révision du plan est quotidien (à l'heure spécifiée par le paramètre '"revisionDuPlan" "heureMatin"') par le jeu de la méthode surchargée DefineNextEventClockTime de la classe d'événements RevisionMatinaleDuPlan.
On rappelle par ailleurs que, lorsqu'une opération se termine, ce n'est pas une révision du plan qui est automatiquement lancée par le moteur de simulation, mais seulement un nouvel établissement des opérations à acter (compte tenu de la libération des ressources de l'opération terminée).
Pour capter l'opportunité d'ouvrir une activité sans attendre le lendemain matin, un examen complémentaire en milieu de journée est programmé si une opération a été terminée dans la première moitié de la journée. Cette tâche est confiée au moniteur sur le degré de progression de l'opération (descripteur hérité AchievementDegree), lorsqu'on passe d'un degré de 1. L'heure au-delà de laquelle une fin d'opération n'entraîne plus l'examen complémentaire est spécifiée par un paramètre dont le premier identifiant est "revisionDuPlan" et le second "heureMiJourneeNormale" ou "heureMiJourneePointe". En particulier, si on ne veut pas exploiter cette possibilité, on donne à ce paramètre la valeur de l'heure de début de journée.

Annexe 5 :

Présentation du module SISOL intégrable au simulateur SILASOL

Jean-Pierre Rellier (INRA/MIA/UBIA Toulouse)

Janvier 2009

Préambule

SILASOL est un simulateur d'une exploitation de grande culture destiné à étudier l'intégration du Pois (sous différents types variétaux) dans les successions de cultures.

Ce simulateur est développé dans l'environnement de modélisation/simulation DIESE. Il contient une réécriture, dans le formalisme DIESE, d'un modèle de croissance et développement du Pois, adjoint de modules 'bilan hydrique' et 'azote sol' : AFISOL.
Dans AFISOL, l'évolution de l'état du système sol/culture est notamment fonction de l'état de tassement du profil de sol, lui-même influencé par les opérations de travail du sol.
L'occurrence des opérations de travail du sol et leurs effets sont simulées à partir d'un modèle de la conduite des cultures, lui aussi intégré à SILASOL.

Ainsi, ce qu'on appelle dans DIESE la « procédure de transition d'état » d'une opération technique fait appel à une fonction 'gestOT' qui renvoie le nouvel état de tassement en fin d'opération, en connaissance de l'état en début d'opération. Ce nouvel état est synthétisé en ce qu'on appelle le « pourcentage de zone tassée ».

L'état initial de tassement est installé par une procédure 'ini'. Ceci devra être fait en début de la période simulée. Mais dans la version courante du simulateur (Silasol.1.1), cela est réalisé dans la procédure de transition d'état de l'unique (par choix de maquettage) opération de labour de l'unique parcelle en culture de Pois. Et bien entendu avant l'appel de l'(unique) appel à 'gestOT'.

La fonction 'gestOT' fait elle-même appel à d'autres procédures ou fonctions (compactage, labour, reprise, ...), et ainsi de suite en profondeur. Le code de cet ensemble de fonctions est issu d'un programme autonome, SISOL, incluant aussi une interface d'utilisation dans un OS Windows (J. Roger-Estrade, H. Beaufils).

Le principe du travail d'intégration de SISOL dans SILASOL a été de détacher le modèle de son interface. Autrement dit, d'extraire de SISOL le code nécessaire et suffisant pour le calcul d'une unique variable de sortie : le pourcentage de zone tassée. Ce code C++ est intégré dans une librairie dynamique (lib_SISOL[.so ou .dll]), elle-même référencée lors de l'édition des liens au moment de produire le simulateur exécutable SILASOL. C'est ainsi que des fonctions telles 'gestOT' peuvent être invoquées dans la « procédure de transition d'état » des opérations techniques.

Par ailleurs, les fonctions de la librairie lib_SISOL peuvent être invoquées en dehors de SILASOL : c'est notamment le cas dans un programme spécialement dédié à la mise au point et au test de la librairie (sisolJPR), et qui va être le support du questionnement posé ci-après.

Dernier élément de préambule, l'intégration de SISOL dans SILASOL a nécessité de revisiter le passage de valeurs entre les fonctions, en termes de liste d'arguments, de valeurs renvoyées et de variables globales. Parfois aussi, des changements de noms variables fonctions) ont été jugées utiles ou nécessaires (par exemple la fonction 'gestOT' est la fonction 'MainProg' du programme original). Enfin, et ce n'est pas le moins important, des structures définies dans DIESE ont été exploitées (notamment la classe générique Tableau), ce qui nécessite d'associer la librairie dynamique de DIESE lors de la génération de l'exécutable de mise au point sisolJPR.

Manipulation

Le programme sisolJPR est livré sous la forme d'une archive compressée, de nom 'release_sisolJPR'. Son contenu est le suivant :

```
release_sisolJPR : {
    include/      : les fichiers d'entête C++
    libLinux/     : la destination des objets résultant de la compilation sous linux
    libCygwin/    : la destination des objets résultant de la compilation sous Windows+Cygwin
    READ_ME       : un mode d'emploi pour l'exécution
    source/       : les fichiers de réalisation C++
}
```

Le fichier READ_ME indique qu'une fois l'exécutable généré, celui-ci est lancé par une ligne de commande dans un terminal. Trois niveaux de trace sont spécifiés par les arguments de la commande. La trace de l'exécution peut être redirigée vers un fichier texte, pour être ensuite éditée. Essentiellement, la trace consiste en l'affichage du profil rectangulaire du sol, chaque cellule de la matrice indiquant si le sol est ou non tassé en ce point {x.y}.

L'utilisateur a un accès ouvert aux « sources » du programme. Il peut notamment ajouter des instructions de trace d'exécution, pour observer plus finement le comportement du programme.

Contenu des fichiers :

main.cpp	: main
gestot.cpp	: gestOT
operat.cpp	: compactage, labour, reprise
struct.cpp	: ini, raz_tab, fini, pourcentDellta, …
templateTableau.cpp	: instanciations de la classe Tableau pour Quadra, Quadra2 et Tableau<Quadra>
utils.cpp	: TriJRE, Tribande, ModifAleatoire, calculeLargeurCompactee, calculeProfondeurCompactee, …

Annexe 6 : analyse du module de résistance au froid

Mise à plat programme AFISOL : MODULE GEL
Rédaction Nathalie Cialdella, Christophe Lecomte, le 07/03/2009

Prise en compte du stress thermique :

Gras souligné, paramètres variétaux à rentrer par l'opérateur

Calcul de la résistance des plantes au froid
(1) **Rmin** : résistance minimale (quand il n'y a aucun endurcissement)
(2) **Rs** : résistance seuil de la variété (résistance qui peut être atteinte quand le stade de la plante, les conditions de température et la durée d'endurcissement sont maximum)
(3) Rmax(j) : résistance maximale permise par le stade de la plantule (Rmax(j) ≤ Rs)
(4) Rpot(j) : résistance potentielle (résistance permise par la température moyenne journalière, qui conditionne l'endurcissement : Rmin ≤ Rpot(j) ≤ Rmax(j))
(5) R(j) : résistance atteinte le jour j, dépendant de la résistance du jour j-1 et de Rpot(j)

Si somme températures depuis semis<0 alors :
Rmax(j)= **Rmin**
Rpot (j) =**Rmin**
R(j) = **Rmin**
Sinon
Si Tmoy (j) ≥ **Tmaxgamme** alors Rpot(j) = **Rmin**
Si R(j-1) = Rpot(j) alors R(j)=R(j-1)=**Rmin**
Sinon si R(j-1)<Rpot(j), alors R(j)= Min ((R(j-1)+Tmoy(j)*(**Rmin**-**Rs**)/(20***nbjdésendurcissement**) ; **Rmin**)
Si **Tmingamme** <Tmoy(j) <**Tmaxgamme** alors Rpot(j)=Rmax(j)+Tmoy(j)*(**Rmin**-Rmax(j))/**Tmaxgamme**
Si R(j-1)>Rpot(j) alors R(j) = Max ((R(j-1)+(Rmax(j)-**Rmin**)/**nbjendurcissement***(1-Tmoy(j)/**Tmaxgamme**)) ; Rpot(j))
Si R(j-1) = Rpot (j) alors R(j) = R(j-1) = Rpot(j)
Si R(j-1)<Rpot(j) alors R(j) = Min((R(j-1)+Tmoy(j)*(**Rmin**-**Rs**)/(20***nbjdésendurcissement**) ;Rpot(j))
Si Tmoy(j) ≤ **Tmingamme** alors Rpot(j)=Rmax(j)
Si R(j-1)>Rpot(j) alors R(j) = Max((R(j-1)+(Rmax(j)-**Rmin**)/**nbjendurcissement***(1-Tmin(j)/**Tmaxgamme**)) ; Rmax(j))
Si R(j-1) = Rpot(j) alors R(j)=R(j-1)=Rmax(j)

Calcul Rmax(j) : (pour un phylloterme de 70 dj)
Si ∑Tmoy(j)semis≤ **180** alors Rmax(j)=**Rmin**
Sinon si ∑Tmoy(j)semis ≤ **210**
Alors Rmax(j)= **Rmin** + ((**Rs**-Rmin)*(∑Tmoy(j)semis - **180**/(**210**-**180**))
Sinon Rmax(j)=**Rs**
180 = sdj à la date de levée (pour un phylloterme de 70 dj) pour Tmingamme = 0
210 = sdj au stade stade ½ feuille (pour un phylloterme de 70 dj) pour Tmaxgamme=10

Tmingamme : T° Minimale en dessous de laquelle l'endurcissement est maximal pour la variété considérée
Tmaxgamme : T° Maximale en dessus de laquelle il n'y a pas d'endurcissement pour la variété

considérée

Effet sur la limitation de la croissance de la plante RT(j) :

si R(j)-Tmin ≤ 0 alors RT(j)=0 (pas de destruction)
si R(j)-Tmin ≥ 10 alors RT (j)=1 (destruction totale)
sinon RT(j)=0.1*(R(j)-Tmin) (l'augmentation des dégâts sur la matière sèche est proportionnelle)

Effet sur l'accumulation de Matière Sèche :
Si RT(j)<1 alors MS(j)= MS(j-1) + (1-RT(j))*ΔMS(j)
Sinon, MS(j)=0 et Rendement = 0 (sans arrêt de la simulation, si possible)

Calcul Stress thermique pendant la période d'initiation florale (Sth) :

Si SomTempMinIF ≤ SomTemp(j)≤SomTempMaxIF
Et Si R(j)>Tmin(j) alors Sth(j)=R(j)-Tmin(j)
Sinon Sth(j)=0

Avec SomTempMinIF : somme de température minimale de déclenchement de l'initiation florale pour la variété considérée
Et SomTempMaxIF : somme de température maximale de déclenchement de l'initiation florale pour la variété considérée

Ensuite, ça serait bien de pouvoir renvoyer la valeur cumulée des Sth(j) en sortie.

Tests à faire avec :
Pour les variétés d'hiver Rmin = -3 ; Rs = -18 ; nb j endurcissement = 42 ; nb j désendurcissement = 5 ; Tmingamme= 0 ; Tmaxgamme = 15

Pour les variétés de printemps, prendre la même chose avec Rs = -10 ; Tmingamme = 0 ; Tmaxgamme = 10.

Relations ci-dessus établies à partir du mode de calcul de la résistance au gel du blé ci dessous : fonction de la température moyenne journalière et de la résistance acquise le jour précédent, dans le cas d'une vitesse d'endurcissement constante et d'une durée d'endurcissement de 28 jours (Lecomte, 2006).

Condition sur la température moyenne journalière	Résistance potentielle au froid (Relation (2))	Condition sur la résistance acquise au jour j-1	Variation de résistance au froid (Relations (3) ou (4))	Résistance au jour j
Si Tm >= 15	$Rpot = Rmin$	Si $R_{j-1} = Rpot$	$\Delta R = 0$	$R_j = R_{j-1} = Rmin$
		Si $R_{j-1} < Rpot$	$\Delta R = Tm * \frac{(Rmin - Rs)}{100}$	$R_j = MIN [(R_{j-1} + \Delta R) ; Rmin]$
Si 0 < Tm < 15	$Rpot = Rmax + Tm^{\frac{1}{5}} * (Rmin - Rmax)$	Si $R_{j-1} > Rpot$	$\Delta R = \frac{(Rmax - Rmin)}{28} * (1 - Tm^{\frac{1}{5}})$	$R_j = MAX [(R_{j-1} + \Delta R) ; Rpot]$
		Si $R_{j-1} = Rpot$	$\Delta R = 0$	$R_j = R_{j-1} = Rpot$
		Si $R_{j-1} < Rpot$	$\Delta R = Tm * \frac{(Rmin - Rs)}{100}$	$R_j = MIN [(R_{j-1} + \Delta R) ; Rpot]$
Si Tm <= 0	$Rpot = Rmax$	Si $R_{j-1} > Rpot$	$\Delta R = \frac{(Rmax - Rmin)}{28} * (1 - Tm^{\frac{1}{5}})$	$R_j = MAX [(R_{j-1} + \Delta R) ; Rmax]$
		Si $R_{j-1} = Rpot$	$\Delta R = 0$	$R_j = R_{j-1} = Rmax$