Représentation et interprétation de plans de production flexibles

Representing and interpreting flexible production plans

Roger Martin-Clouaire Jean-Pierre Rellier

INRA

Unité de Biométrie et d'Intelligence Artificielle –UR875
BP52627 Auzeville, 31326 Castanet Tolosan, France
rmc@toulouse.inra.fr, rellier@toulouse.inra.fr

Résumé

En agriculture, la difficulté du problème de conduite de production provient du rôle de facteurs incontrôlables comme le climat sur les processus biophysiques sous-jacents à toute production agricole. Pour faire face à cette incertitude le comportement décisionnel doit s'appuyer sur une sorte de plan flexible qui permet de retarder la détermination des actions jusqu'à ce que les faits pertinents soient connus.

L'article présente un langage dédié de représentation de plan qui autorise une spécification des contraintes sur et entre les activités. Un interpréteur opère sur des plans construits manuellement et détermine itérativement les activités qu'il est licite d'exécuter selon la spécification. Une fois représenté dans ce cadre, un plan peut être simulé dans différents scénarios de facteurs externes, ce qui permet d'étudier et de construire empiriquement des comportements décisionnels de conduite.

Mots Clef

Interprétation de plan, activités, opérations, événements.

Abstract

In agriculture the production management difficulties stem from the high influence that uncontrollable factors like weather or pests have on the biophysical processes underlying any agricultural production. In order to cope with uncertainty the decision making behavior must rely on a kind of flexible plan that enables to postpone the full determination of actions until execution time.

The paper presents a dedicated plan representation language that supports the specification of a well structured set of intended activities and an interpreter that takes as input this handcrafted plan and determines repeatedly over time the activities that are currently eligible for execution. Once represented in this framework a production management plan can be simulated in various exogenous conditions, which enables the empirical study and design of management behaviors.

Keywords

Plan interpretation, activities, operations, events.

1 Introduction

Cet article présente un langage de représentation de plan conçu pour étudier par simulation des processus de gestion de production qui sont fortement dépendants de facteurs exogènes incontrôlables. L'approche est motivée par des applications à des systèmes de production agricoles (systèmes d'élevage ou productions végétales). Les rendements, qualité et coûts des productions agricoles sont fortement affectés par les conditions climatiques, les maladies, les ravageurs ou d'autres causes incertaines. La profitabilité et la viabilité économique d'une exploitation agricole dépendent des procédures mises en place pour effectuer en temps utile et efficacement les interventions agronomiques requises et limiter les risques ou les conséquences des événements défavorables.

La nature biologique des productions agricoles les rend fondamentalement différentes des productions manufacturières. L'incertitude dans l'industrie concerne principalement les objectifs de production (la demande) et, à un degré moindre, la disponibilité de ressources (e.g. panne de machines). En agriculture, l'incertitude affecte le déterminisme des actions et force l'enchaînement des actions à être dépendent du contexte pour faire face à des menaces ou saisir des opportunités. Les processus de production manufacturée sont entièrement conçus par l'homme et sont de ce fait pensés en cohérence avec les besoins de planification et d'ordonnancement des opérations. En agriculture les processus de production peuvent se poursuivre sans intervention humaine sous l'effet des facteurs moteurs naturels (lumière, énergie). De plus, les critères de performance diffèrent ; dans l'industrie on cherchera typiquement à réduire les temps d'exécution alors qu'en agriculture on s'attachera en priorité à la maîtrise des risques liés aux facteurs incontrôlables.

En dépit de l'incertitude et de la variabilité inhérentes aux systèmes biophysiques les processus de décision en conduite de production agricole ne sont toutefois pas purement réactifs. Des structures et des régularités d'une année à l'autre permettent aux agriculteurs de planifier dans les grandes lignes les principales activités requises par un objectif de production dans un contexte donné.

Cet article présente un cadre permettant de spécifier l'organisation temporelle flexible des activités et de déterminer au fur et à mesure quelles actions il est licite d'exécuter selon le plan ainsi spécifié et compte tenu des circonstances rencontrées. Le but associé à cette modélisation est de pouvoir étudier les comportements décisionnels en conduite de production agricole et de réaliser de l'expérimentation virtuelle de ces comportements par simulation sur ordinateur. Pouvoir expliciter et simuler des comportements décisionnels est un moyen de faciliter la communication, l'apprentissage et la conception de façon de conduire une production.

La section suivante présente brièvement les systèmes de production agricoles et la tâche de conduite de production. Les bases du langage de représentation de plan sont explicitées en Section 3. L'algorithme qui gère l'état des activités composant le plan est donné en Section 4. La section 5 situe la contribution de l'article relativement à des travaux connexes.

2 Systèmes de production agricoles

Un système de production agricole (voir Figure 1 qui illustre un système de production sous serre) est vu comme une entité située dans ce qui est appelé l'environnement extérieur (e.g. le contexte climatique et économique) et peut être décomposé en 3 sous-systèmes interactifs : le pilote, le système opérant et l'appareil producteur. Un système de production est une entité active dans le sens qu'il est le siège de processus, a des entrées (physiques ou informationnelles), des sorties, et est soumis à un agenda d'événements. Les événements (flèches pleines) contrôlent (activent) les processus.

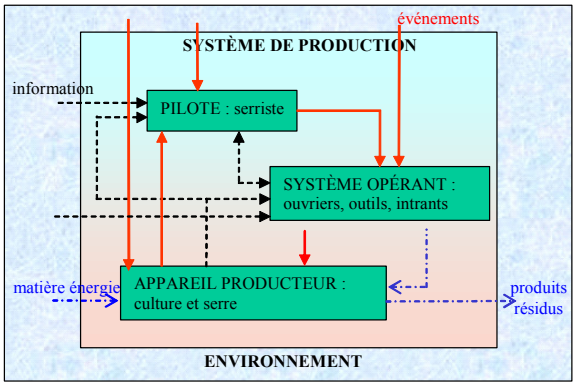

Figure 1. Système de production agricole

L'appareil producteur, aussi appelé système biophysique, est composé d'entités biophysiques (e.g. culture, animaux, milieu physique) qui ont leurs propres processus (e.g. photosynthèse, croissance, physiologie). Parmi les événements contrôlant ces processus se trouvent ceux activés par l'exécution des opérations réalisées par le système opérant. Les entrées sont les intrants (e.g. engrais ou pesticides provenant du système opérant) et l'énergie issue de l'environnement extérieur ou fournie par le système opérant. Les processus peuvent générer des événements particuliers liés à des changements notables de l'état du système biophysique qui peut être équipé de capteurs et de dispositifs d'alarme.

Le pilote est le décideur (agriculteur) qui a la responsabilité de réaliser les finalités du système de production. Dans notre modèle, le pilote est doté d'une stratégie de conduite qui régit le comportement du système opérant et, indirectement, oriente celui du système biophysique. La stratégie de conduite que se donne l'agriculteur spécifie dans un plan la façon flexible d'organiser les activités, les adaptations qu'elle doit subir lorsque certains événements se produisent, et les principes à respecter par le système opérant pour établir précisément les actions à exécuter. Elle définit aussi le rythme auquel le pilote doit l'examiner.

Comme le processus de production est fortement influencé par des facteurs incontrôlables, l'agriculteur doit veiller à la robustesse de sa stratégie. Elle doit convenir dans presque tous les scénarios climatiques et répondre correctement aux contingences envisageables ; les effets néfastes peuvent être éliminés ou atténués par des pratiques agronomiques articulant dans le temps les activités appropriées. Par conséquent, le comportement décisionnel de l'agriculteur doit à la fois s'appuyer sur des plans et être réactif aux circonstances rencontrées.

Les capacités de planification des agriculteurs varient d'un individu à l'autre et résultent de l'expérience acquise et de l'expertise transmise par les structures de conseil. Les stratégies de conduite qu'ils se donnent visent à mettre en cohérence leurs intentions d'action avec l'objectif global de l'entreprise, les limitations de moyens, ainsi que leur propre perception et compréhension du fonctionnement de leur système de production et des événements pertinents à surveiller.

Les processus attachés au pilote portent sur :

- la surveillance de l'occurrence d'événements décisifs et l'inspection de l'appareil producteur sur les aspects importants pour la conduite ;
- la mise à jour des activités en fonction de l'avancée du temps et des changements survenus (e.g. certaines activités sont devenus obsolètes alors que d'autres sont devenues potentiellement exécutables);
- la révision de la stratégie dans des situations connues pour l'exiger ;
- l'extraction par lecture de la stratégie des jeux d'activités candidats à exécution et la sollicitation du système opérant pour qu'il traite ces jeux.

Chaque fois que le pilote est activé il produit un ensemble de jeux d'activités dont l'exécution est licite selon le plan, et les transmet au système opérant qui doit dériver celles qui sont réellement à exécuter étant donné les ressources disponibles et les préférences énoncées. Le système opérant utilise sa propre procédure de résolution dans le cadre de l'autonomie qui lui est fixée. Il est doté pour cela de processus permettant de :

- rechercher une ou des allocations de ressources aux activités ;

- sélectionner le meilleur jeu d'activités en cas de concurrence entre plusieurs jeux.

L'exécution par le système opérant du jeu qu'il a sélectionné se poursuit jusqu'à ce qu'un changement sur les ressources survienne (fin d'une activité ou fin de l'horaire de travail). Un tel événement provoque un nouveau calcul de jeu à exécuter ou un passage du contrôle au pilote qui poursuit sa tâche tant qu'il reste des activités en attente d'exécution.

3. Représentation de plans

3.1 Activités

Au centre de la problématique de conduite se trouve le concept d'*activité*. Dans sa forme la plus simple, une activité, alors qualifiée de *primitive*, spécifie une opération (récolter les fruits mûrs) à faire sur une entité biophysique (une plante) ou un emplacement (une parcelle) par un exécutant (un ouvrier, un robot ou un ensemble de ces entités). Outre ces trois composantes, une activité est caractérisée par des conditions d'ouverture ou de fermeture définies par des fenêtres temporelles (dates au plus tôt et au plus tard) et/ou des prédicats relatifs à un état (e.g. atteinte d'un stade physiologique, disponibilité d'une ressource). Ces conditions servent à déterminer les activités dont l'exécution est licite selon le plan ; elles expriment typiquement une pertinence agronomique plutôt qu'une impossibilité matérielle. Toute activité, primitive ou non, a un état dont les valeurs possibles sont : *sleeping*, *waiting*, *open*, *closed* ou *cancelled* (leur signification est donnée dans la suite).

Une opération est une transformation intentionnelle qui lors de son exécution cause des changements du système biophysique autre que ceux dus à sa dynamique propre. Ces effets sont instantanés ou sont progressifs sur une période. Une opération affecte soit des individus dans une collection d'objets (e.g. des plantes dans une population) soit des objets possédant des descripteurs numériques (e.g. surface). La vitesse d'une opération est définie par la quantité (e.g. nombre d'items, surface) traitable en une unité de temps. Sa durée est le rapport de la quantité totale par sa vitesse. La faisabilité d'une opération peut être conditionnée par un prédicat[1] portant sur l'état du système biophysique (e.g. humidité du sol pour permettre l'accès des engins).

Les activités dont il a été question jusqu'ici sont primitives. Elles peuvent faire l'objet de contraintes supplémentaires en prenant en compte des relations temporelles de séquencement et de synchronisation entre elles, et en utilisant des structures de programmation spécifiant des choix entre activités alternatives, des itérations d'une activité, le regroupement d'activités ou encore le caractère optionnel d'une activité. A cette fin sont utilisés des opérateurs tels que *before*, *meet*, *overlap*, *co-start*, *co-end*, *equal*, *or*, *and*, *iterate* et *optional* qui, appliqués à des activités primitives, donnent naissance à d'autres activités dites non-primitives. Plus généralement, un opérateur de composition appliqué à des activités (primitives ou non) qui sont ses arguments définit une nouvelle activité ; on dira des activités arguments qu'elles sont les filles de l'activité composée. Une activité composée peut recevoir des conditions d'ouverture et de fermeture comme à une activité primitive. L'ouverture et la fermeture d'une activité[2] construite par composition à l'aide d'un opérateur sont induites par les conditions d'ouverture et de fermeture des activités primitives sous-jacentes et par la sémantique des opérateurs de composition. Toutes les activités sont connectées ; la seule qui n'a pas de mère est appelée *plan nominal*.

Ce plan est flexible dans le sens que deux séquences différentes d'événements vont vraisemblablement amener deux réalisations différentes de ce plan. Les dates d'ouverture diffèreront par exemple. De plus certaine activités pourront être annulées dans un cas et pas dans l'autre si elles sont optionnelles ou sujettes à un choix dépendant du contexte.

L'avancée du temps et le changement d'état du système de production est susceptible de rendre satisfaites les conditions qui régissent le changement d'état des activités. Le changement d'état des activités se produit à des moments particuliers spécifiés par le pilote (rythme d'examen du plan nominal) et lorsqu'une opération se termine. Tout changement d'état d'une activité est propagé aux activités auxquelles elle est directement ou indirectement connectée par les opérateurs de composition.

La signification des états possibles des activités peut être expliquée maintenant. La valeur *sleeping* est donnée à chaque activité lors de sa création. Elle signifie que les conditions d'ouverture et de fermeture n'ont pas encore besoin d'être surveillées. L'état passe à *waiting* aussitôt que les conditions d'ouverture de l'activité doivent être examinées. Par exemple, dès qu'une activité se termine (et pas avant), il convient de surveiller les conditions d'ouverture de celle qui la suit dans une séquence spécifiée avec l'opérateur *before*. Le plan nominal en tant qu'activité est déclaré *waiting* à l'initialisation de la simulation. L'état d'une activité passe à la valeur *open* lorsque ses conditions d'ouverture sont satisfaites[3]. L'état passe de la valeur *open* à la valeur *closed* lorsque les conditions de fermeture de l'activité sont satisfaites, ou bien, s'il s'agit d'une activité primitive, lorsque l'opération

1 Une condition de faisabilité est propre à l'opération mais rien n'empêche d'utiliser la même condition pour contraindre l'ouverture d'une activité portant sur cette opération.

2 Pour être exact, il faudrait parler de spécification d'activité plutôt que d'activité pour garder à ce dernier terme le sens d'intervention exécutée alors qu'il s'agit ici de tâche intentionnelle incomplètement déterminée et éventuellement jamais exécutée (cas d'utilisation dans les opérateurs *or* et *optional*).

3 Le passage à l'état open d'une activité primitive ne signifie pas que son exécution va démarrer sur le champ. Cette décision dépend d'autres conditions (faisabilité de l'opération sous-jacente, disponibilité de ressources) et d'un éventuel arbitrage.

en jeu est terminée. L'état d'une activité passe à la valeur *cancelled* lorsqu'il n'y a plus lieu d'opérer un quelconque changement d'état. Typiquement, cela se produit lors d'un choix entre deux ou plusieurs activités alternatives (opérateur *or*) : l'état des activités non choisies passe à *cancelled*.

La sémantique de chaque opérateur de composition est définie par deux ensembles de règles spécifiant :

- les pré-conditions que doit satisfaire l'activité mère pour autoriser le changement d'état de certaines activités filles et vice versa;
- les post-conditions ou effets d'un changement d'état des activités filles sur la mère et vice versa.

Les cas des opérateurs *before*, *iterate*, and *optional* sont détaillés dans les sous-sections suivantes. Voir [5] pour une présentation exhaustive.

3.2 Contraintes de séquencement

Pour spécifier que deux ou plus de deux activités doivent se suivre sans recouvrement lors de leurs exécutions respectives on connecte ses activités par l'opérateur *before*. Par exemple dans le cas de 3 activités A, B, C, l'activité composée *before*(A B C) impose que B ne peut pas être dans l'état *open* avant que A soit dans l'état *closed*. L'ordre de la séquence est exprimé par l'ordre des arguments de l'opérateur. Toute activité composée à partir de *before* a deux propriétés supplémentaires qui permettent de spécifier si besoin des délais entre les ouvertures d'activités consécutives et des délais entre fermeture et ouverture d'activités consécutives.

Les pré-conditions à un changement d'état sont les suivantes. Pour que l'état de la mère puisse passer :

- *waiting* (resp. *open*), sa première fille doit pouvoir passer *waiting* (resp. *open*);
- *closed*, sa première fille doit pouvoir passer *closed*.

Pour que la première fille puisse devenir :

- *waiting* (resp. *open*), la mère doit pouvoir passer *waiting* (resp. *open*).

Pour qu'une fille autre que la première puisse passer :

- *waiting*, la précédente doit être *closed* ou autorisée à le devenir.

Pour que la dernière fille puisse passer :

- *closed*, la mère doit pouvoir passer *closed*.

L'effet du changement d'état de la mère ou des filles se traduit par les règles suivantes. Aussitôt que la mère devient :

- *waiting* (resp. *open*), la première fille devient *waiting* (resp. *open*);
- *closed*, la dernière fille devient *closed*.

Aussitôt qu'une fille devient :

- *waiting* et qu'elle est la première fille alors la mère devient *waiting*. Sinon, la fille précédente devient *closed* (si ce n'est pas déjà le cas);
- *open* et qu'elle est la première fille alors la mère devient *open*;
- *closed* et qu'elle est la dernière fille alors la mère devient *closed*. Sinon, la fille suivante devient *waiting* si possible.

L'opérateur *meet* est un autre opérateur de séquencement. Il est très similaire à *before* sauf qu'il impose la synchronisation de la fermeture d'une fille avec l'ouverture de la suivante.

3.3 Itération

L'opérateur *iterate*, qui a un seul argument, spécifie que l'activité fille doit être répétée dans l'intervalle de temps pendant lequel l'activité mère est *open*. La mère doit impérativement avoir être dotée de conditions d'ouverture et de fermeture (intervalles de date ou prédicats) ou des nombres maximum ou minimum d'itérations ou encore de combinaisons de ces possibilités. La fille ou plus généralement les activités descendantes au sein de l'argument ne doivent pas apparaître ailleurs dans le plan. La mère a deux propriétés supplémentaires permettant de spécifier si besoin les délais entre l'ouverture de deux itérations consécutives de la fille et entre la fermeture d'une itération et l'ouverture de la suivante. Les seules pré-conditions au changement d'état de la fille sont que la mère soit *waiting* ou *open* pour que la fille devienne *waiting*, et que la mère soit *open* pour que la fille puisse passer *open* ou *closed*.

Dès que la mère devient :

- *open*, la fille devient *waiting* si possible;
- *closed*, la fille devient *closed* si possible.

Dès que la fille devient *closed*, elle passe immédiatement à *waiting* sauf si les conditions de fermeture de la mère sont satisfaites à ce moment.

Le processus d'itération, qui est contrôlé par une procédure spécifique, duplique (instancie en fait) l'activité fille en accord avec les contraintes spécifiées sur les délais et nombres d'itération. Ces copies ont un état changeant de *sleeping* à *waiting*, de *waiting* à *open*, de *open* to *closed*, et, exclusivement pour ce cas, de *closed* à *waiting*. Ces transitions continuent tant que la mère est *open*.

3.4 Activité optionnelle

L'opérateur *optional*, appliqué à une activité qui est son unique argument, exprime que si elle ne peut pas être réalisée (i.e. il est trop tard relativement à l'intervalle de ouverture) alors cet empêchement ne constitue pas une raison suffisante pour déclarer le plan non-viable. En d'autre terme, cet opérateur spécifie que l'activité fille est à réaliser si possible. La fille ou plus généralement les activités descendantes au sein de l'argument ne doivent pas apparaître ailleurs dans le plan. L'état de la mère peut devenir *waiting* si la fille peut devenir *waiting*. Dès pré-conditions analogues tiennent si on substitue *waiting* par *open* ou par *closed* et aussi en permutant fille et mère. Les règles régissant les effets suivent les règles de pré-conditions (e.g. la fille devient *open* dès que la mère devient *open*). Lorsqu'une activité mère faite avec

l'opérateur *optional* ne peut être réalisée, son état est forcé à la valeur *closed*.

4. Mise à jour de l'état des activités

4.1 Algorithme

L'avancée du temps et le changement d'état du système de production est susceptible de rendre satisfaites les conditions qui régissent le changement d'état des activités. Le changement d'état d'une activité est réalisé par une procédure qui, essentiellement, vérifie que les conditions d'ouverture et/ou de fermeture peuvent être satisfaites et que les contraintes liant cette activité à d'autres seraient satisfaites si le changement était fait. Cette procédure, appliquée au plan, cause un examen récursif de toutes les activités dans l'état *waiting* ou *open*. Toute activité dont le changement est validé est mise à jour et le changement est propagé immédiatement aux activités qui lui sont liées.

Une présentation plus formelle de ce processus est donnée à travers le pseudo-code de la procédure Update qui suit.

```
procedure: Update(activity)
    if      activity.status not waiting and
            activity.status not open
    then    return
/* début de détection d'échec au plan */
    if      {activity.status = waiting and
            it is no longer possible to open} or
            { activity.type = primitive and
            activity.status = open and
            opening time is over and
            operation is not yet executing}
    then    if   activity.type = optional
            then {TurnToClosed(activity); return}
            else  exit("Plan failure")
    if      activity.status = open and
            it is no longer possible to close
    then exit("Plan failure")
/* fin de détection d'échec au plan */
    switch activity.type
            case primitive
                 if      ?OpeningValid(activity)
                 then    TurnToOpen(activity)
                 if      ?ClosingValid(activity)
                 then    TurnToClosed(activity)
                /* ?ClosingValid inclut 1 test relatif  au pourcentage
             minimal de réalisation de l'exécution, typiquement 100%*/
            case iteration
                 if      ?OpeningValid(activity)
                 then  TurnToOpen(activity)
                 if      status = open
                 then switch child.status
                          case sleeping
                               TurnToClosed(activity)
                          case waiting
                             if   ?ClosingValid(activity)
                             then TurnToClosed(activity)
            case others
                 for     each child do Update(child)
```

Normalement la procédure Update est invoquée répétitivement jusqu'à ce que le plan devienne *closed*. Dans certains cas, le plan ne peut plus être fermé, ce qui révèle un échec au plan. Une telle situation se produit lorsque certaines pré-conditions au changement ne peuvent plus être satisfaites (e.g. une activité *meet* dans laquelle la seconde fille ne peut être ouverte alors que la première doit être fermée). En d'autres termes, ceci survient lorsqu'une activité non-optionnelle ne peut plus être ouverte ou lorsqu'elle ne peut plus être fermée sans violer des contraintes qui la lient à d'autres activités par les opérateurs de composition.

Deux prédicats importants sont utilisés dans Update : ?OpeningValid et ?ClosingValid. Ils retournent *true* s'il est licite d'ouvrir ou fermer l'activité passée en argument. ?OpeningValid appelle deux prédicats dépendent du type d'activité : ?CheckSonsIfOpen et ?CheckIfSonOpen. Ceux-ci avec les prédicats ?CheckSonsIfWaiting, ?CheckIfSonWaiting, ?CheckSonsIfClosed, et ?CheckIfSonClosed sont la traduction des pré-conditions au changement définies pour chaque opérateur. Ils appellent eux-mêmes ?OpeningValid, ?ClosingValid et ?WaitingValid. Dans le principe, ces 3 prédicats sont très similaires. Le pseudo-code de ?OpeningValid est donné ci-dessous. Pour la clarté, ce code a été épuré de toutes les structures et de tous les tests permettant d'éviter les boucles.

```
predicate: ?OpeningValid(activity)
  if  activity.status = open  then  return  true
  else
      if  {activity.status = waiting or ?WaitingValid(activity)}
          and local opening conditions satisfied
      then
          if ?CheckSonsIfOpen(activity)
          then  {for each mother do
                    if   not ?CkeckIfSonOpen(activity, mother)
                    then     return false}; return true
          else  return false
       else return false
```

Update appelle TurnToOpen et TurnToClosed. Elles réalisent avec TurnToWaiting les changements d'état correspondants sur l'activité passée en argument et propagent l'effet aux activités connectées. Une fois invoquées (soit dans Update ou au début de la simulation quand le plan est forcé à passer de *sleeping* à *waiting*) elles font les mises à jour selon les règles définies pour chaque opérateur. Notons que les prédicats ?OpeningValid, ?ClosingValid et ?WaitingValid sont aussi utilisés dans ces procédures de réalisation de changement d'état.

4.2 Exemple

Les concepts et mécanismes définis dans les sections précédentes ont été utilisés pour décrire un système de production de tomates sous serre [3]. Pour illustrer cette application, on considère ici un plan extrêmement simplifié qui est en fait une petite partie d'un plan réel pour ce type de production. Le plan est le suivant :

before(iterate(ECLAIRCISSAGE1), iterate(optional(ECLAIRCISSAGE2)))

Il spécifie que deux séries d'activités d'éclaircissage qui doivent être faites successivement et que l'éclaircissage de

la seconde série est optionnel. Les activités ECLAIRCISSAGE1 et ECLAIRCISSAGE2 sont toutes deux primitives et consistent en l'application de l'opération Eclaircir aux plantes d'un compartiment de serre. Cette opération a pour effet d'éliminer des jeunes fruits du bouquet le plus récent de façon à n'en laisser qu'un nombre restreint et éviter ainsi de produire des fruits de trop petit calibre. Les deux activités d'éclaircissage diffèrent seulement par les ressources qu'elles requièrent : la première nécessite un ouvrier qualifié alors que la seconde nécessite aussi un ouvrier mais qui peut être un saisonnier. Soit w1 et w2 deux ouvriers de chacune de ces catégories. Supposons que w1 est disponible du jour 0 au jour 30 alors que w2 a un contrat de travail du jour 30 jusqu'à la fin de la saison et peut être indisponible de temps en temps aléatoirement pour d'autres tâches jugées prioritaires. Ainsi il peut être absent 6 jours d'affilée par quinzaine mais doit être présent au moins cinq jours consécutifs lorsqu'il revient. La surface d'un compartiment de serre est égale à 10 unités et la vitesse d'éclaircissage d'un ouvrier est de 2 unités par jour.

Les spécifications temporelles dans les différentes activités sont exprimées sur une échelle journalière. Le plan lui-même (i.e. l'activité *before*) a des fenêtres d'ouverture et de fermeture égales à [0, 60] et [60, 60] respectivement. La fenêtre d'ouverture de l'éclaircissage de la première série est égale à [0, 5]. Quand un éclaircissage passe à *open* au temps t, la fenêtre d'ouverture de l'itération potentielle suivante dans une série est fixée à [t+10, t+15]. Toute activité éclaircissage a un prédicat de fermeture interdisant qu'elle se fasse plus de 10 jours après le début de l'exécution de l'opération sous-jacente. Les 2 arguments du *before* ont [0, ∞] comme fenêtre d'ouverture ; leurs fenêtres de fermeture respectives sont [30, 60] et [60, ∞]. Enfin le *before* est spécifié de manière que la fenêtre d'ouverture de la première itération de la seconde série vaut [t+10, t+15] où t est la date d'ouverture de la dernière itération dans la première série.

Comme la disponibilité de w2 est aléatoire, le résultat de l'exécution du plan est aussi aléatoire. Une réalisation possible est montrée par la Figure 2 à la fin de l'article. La première série comporte 3 itérations d'éclaircissage qui sont ouvertes aussitôt que possible relativement aux contraintes de délai (aux dates 0, 10 et 20). Elles ne sont jamais interrompues pour cause d'indisponibilité de ressource et l'exécution s'étend sur 5 jours exactement. La première itération de la seconde série se comporte de manière similaire pour les mêmes raisons. A la date 40 un autre éclaircissage s'ouvre mais l'opération ne peut pas être exécutée, w2 n'étant pas disponible. Comme w2 ne revient qu'à la date 46 et qu'une opération Eclaircir ne peut pas commencer à s'exécuter plus de 15 jours après la date d'ouverture de l'itération précédente, cette activité optionnelle ne peut pas être exécutée et est fermée. L'activité candidate suivante est ouverte à la date 50 (i.e. 10 jours après l'ouverture précédente). L'opération s'exécute les jours 51 et 52 (pendant lesquels w2 est disponible) mais cela ne correspond qu'à 40% de la surface à traiter. L'exécution est alors interrompue et reprend dès le retour de w2 à la date 58. L'opération se termine à la date 60, ce qui ne viole pas le prédicat de fermeture d'un éclaircissage. Le plan est fermé à la date 60.

5. Travaux connexes

Plusieurs approches de spécification du comportement d'agent ont été publiées ces dernières années dans la littérature d'IA et de robotique. Les langages d'agent à base de formalismes en logique du premier ordre comme ceux de la famille Golog/ConGolog [1] ont été conçus principalement pour faire du raisonnement formel sur les activités courantes et potentielles d'un agent de façon à s'assurer que certaines propriétés sont bien vérifiées. Ils s'appuient sur des représentations symboliques de l'environnement de l'agent et sur des théories de l'action qui permettent d'exprimer des relations entre des propriétés (*fluents*) dépendantes de situations, l'effet des actions sur ces propriétés et de raisonner sur ceux-ci. ConGolog permet de spécifier des plans complexes qui sont des sortes de procédures de conduite. Il utilise des structures de programmation autorisant l'expression de séquencement, exécution en parallèle, des actions de test, le choix non-déterministe d'actions et l'itération d'action. Un programme ConGolog utilise une version étendue du calcul de situation pour simuler les changements du monde de façon à pouvoir vérifier l'exécutabilité d'une action par anticipation (avant son exécution réelle) et pour choisir intelligemment la bonne branche en présence d'actions alternatives. Un programme ConGolog est capable de tester hors ligne s'il est correct relativement à un but. La séquence d'actions générées comme une trace de la vérification peut être exécutée pour atteindre ce but.

La principale différence avec notre approche est que notre interpréteur ne peut déterminer que progressivement les actions potentiellement exécutables. La non-exécutabilité est une propriété qui n'est révélée que dès qu'une impasse est découverte par la procédure Update. En fait, pour notre domaine d'application, nous recherchons plutôt une estimation statistique de la non-exécutabilité ; un plan qui ne marche pas (ou pas bien) dans des scénarios climatiques extrêmes (e.g. sécheresse sévère) ne sera pas nécessairement rejeté en agriculture. Une situation de non-exécutabilité d'un plan révélée par la simulation appelle une modification du plan ou des ajustements conditionnels dont le rôle est précisément de changer le plan lorsque les circonstances rencontrées dévient trop du cas nominal. D'autre part, nos problèmes de conduite sont riches en contraintes temporelles sur et entre les activités. Nous avons ainsi accordé de l'importance à l'intelligibilité du langage de représentation de plan. Enfin, les actions ont, en agriculture, des conséquences complexes et hautement incertaines qui paraissent difficilement incorporables dans une théorie de l'action permettant d'anticiper précisément

leurs effets. Il n'est pas encore clair que ces approches formelles puissent s'appliquer à des problèmes de cette complexité et portant sur des mondes changeant continument (voir [4] toutefois).

Les cadres traitant de plans réactifs (voir SPARK [6] pour un des plus récents, un membre de la famille PRS [2]) sont aussi connexes au travail présenté dans cet article car ils proposent également des langages pour exprimer une organisation procédurale des actions. Ils sont équipés de mécanismes d'exécution capables de mettre en œuvre des comportements réactifs de prise de décision. Ces langages n'offrent pas la richesse d'expression souhaitable pour les contraintes temporelles sur les activités. En effet un agent PRS est doté d'un ensemble de procédures prédéfinies (appelées plans ou *knowledge areas*) qui spécifie comment réaliser un but. Chaque plan indique quand son utilisation doit et peut être envisagée, et un ensemble d'actions qui spécifient séquentiellement comment réaliser un but ou réagir à un événement. Les capacités de raisonnement temporel sont assez limitées et la possibilité de maintenir une sorte de continuité dans l'application d'un plan semble difficile à reproduire.

Le type de contraintes temporelles flexibles utilisées dans notre représentation de plan est aussi présent dans le système COMIREM [9] qui met en avant un paradigme de planification interactive opportuniste. Dans ce système, l'allocation de ressources est incrémentale et se fait lorsqu'arrivent de nouvelles contraintes de disponibilité et lorsque de nouvelles activités sont incorporées au plan. Il nous semble intéressant d'approfondir la connaissance de ce système.

6. Conclusion

Nous avons présenté un langage de représentation de plan spécialement conçu pour la conduite de systèmes de production qui sont fortement dépendants de facteurs incontrôlables et pour lesquels les activités à réaliser peuvent néanmoins se planifier si on reste flexible sur la spécification des contraintes temporelles entre elles. Ce cadre de représentation [5] et le moteur à événements discrets [8] qui gère les différents processus sont implémentés sous la forme d'une bibliothèque C++.

Comme signalé dans la section précédente, la problématique de conception de systèmes d'exécution mettant en œuvre des stratégies d'actions pré-programmées dans un monde ouvert est aussi abordée par les communautés de planification/ordonnancement et des agents intelligents en intelligence artificielle. Dans ces approches l'accent est mis sur la synthèse automatique de plan, sur la vérification formelle de propriétés souhaitables d'un plan, et sur les performances d'exécution. Parce que nous nous attachons seulement à éliciter et simuler des comportements décisionnels nous donnons une plus grande importance à la richesse expressive du langage de représentation pour qu'il puisse capter les pratiques observées en conduite de production. Le langage doit être suffisamment flexible, pour éviter de forcer un engagement trop précoce et permettre d'entrelacer en temps utile l'interprétation et la révision de plan avec l'allocation de ressources. La synthèse automatique de plan nous paraît encore hors d'atteinte (d'une complexité combinatoire trop grande) pour notre domaine d'application. De ce fait, nous avons choisi avec la simulation une approche de conception empirique (par expérimentation simulée) dans laquelle une partie de la connaissance (en particulier, l'explicitation des moyens possibles d'atteindre des objectifs et des critères d'évaluation globale) n'a pas à être modélisée dans le simulateur.

Comme mentionné en Section 2, faire face à incertitude en conduite de production agricole nécessite une capacité de révision de plan en réaction à des événements particuliers. La façon de le faire n'a pas été explicitée dans cet article comme n'a pas été exposée la représentation des ressources et la procédure d'allocation de ressources. C'est, avec les applications en cours, l'objet d'articles à venir.

A ce stade, le comportement décisionnel modélisé dans notre cadre de représentation ne comporte pas de représentation explicite des buts. Le besoin de le faire se fait ressentir lorsqu'on veut modéliser des comportements anticipatoires. Nous envisageons pour cela une extension du cadre dans l'esprit des approches de type BDI (*Belief, Desire, Intention*) [7]).

Bibliographie

[1] De Giacomo G., Y. Lespérance, H. Levesque, Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121: 109-169, 2000.

[2] Ingrand F., M. Georgeff, and A. Rao, An architecture for real-time reasoning and system control. IEEE Expert, Knowledge-Based Diagnosis in Process Engineering, 7(6): 34-44, 1992.

[3] Jeannequin B., R. Martin-Clouaire, M. Navarrete, J.-P. Rellier. Modeling management strategies for greenhouse tomato production. Proc. of CIOSTA-CIGRV Congress, Turin, 506-513, 2003.

[4] Martin Y. The concurrent continuous FLUX. Proc of IJCAI, Acapulco, 2003.

[5] Martin-Clouaire R., J.-P. Rellier, Fondements ontologiques des systèmes pilotés". Rapport Interne UBIA-INRA, Toulouse-Auzeville. 2004.

[6] Morley D., K. Myers. The SPARK agent framework. Proc. of AAMAS-04, New York, 712-719, 2004.

[7] Rao A., M. Georgeff. BDI agent: from theory to practice. Proc of Int. Conf. on Multiagent systems, San Francisco, 1995.

[8] Rellier J.-P. DIESE : un outil de modélisation et de simulation de systèmes d'intérêt agronomique.

Rapport interne n° 2005/01 – INRA/UBIA Toulouse, 2005.

[9] Smith S.F., D.W. Hildum, D.R. Crimm, COMIREM: an intelligent form for resource management. IEEE Intelligent Systems, 20(2): 16-24, 2005.

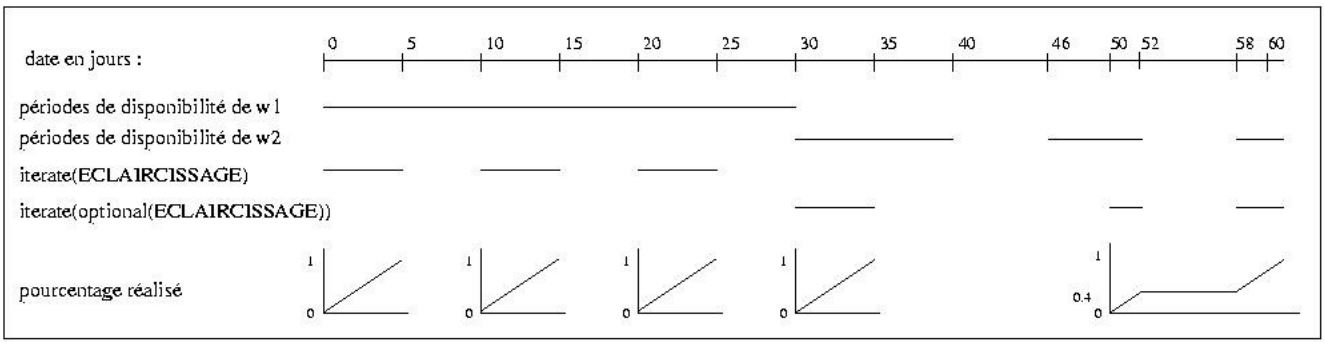

Figure 2. Interprétation du plan compte tenu des disponibilités des ressources w1 et w2