International Congress on Environmental Modelling and Software

2nd International Congress on Environmental Modelling and Software - Osnabrück, Germany - June 2004

Jul 1st, 12:00 AM

Modeling Flexible Plans for Agricultural Production Management

Roger Martin-Clouaire

JP Rellier

Modeling Flexible Plans for Agricultural Production Management

R. Martin-Clouaire and **J.-P. Rellier**

Unité de Biométrie et Intelligence Artificielle, INRA, BP27 Auzeville, 31326 Castanet Tolosan, France.

Abstract: Analyzing in what circumstance and why an agricultural production system performs acceptably well or fails requires models of management practices and work processes. A farmer's management task relies on planned and reactive behaviors that enable him to organize his work in function of known and exploitable regularities, and to adapt it to uncontrollable contingencies as they occur. Consequently, a farm production manager exhibits a decision-making behavior that seems to rely heavily on a kind of flexible plan. This paper aims at presenting a generic model of such plans that are similar to programs. Once represented in this framework a production management plan can be simulated in various exogenous conditions, which enables the study of the underlying production management behavior.

Keywords: Farm production management; Plan; Activity; Time; Event; Simulation

1.   INTRODUCTION

Farm management consists in making and implementing the decisions involved in organizing and controlling a farm enterprise toward an objective that integrates socio-economic and environmental concerns. This paper focuses on production management that deals with the farmers' active role of manipulating the underlying biophysical system (crops and/or livestock) through technical operations. A farm manager must have beforehand an idea of how he intends to get where he wants his production system to move. The global management behavior can be seen as the result of the dynamic interpretation of his management strategy.

A strategy [Martin-Clouaire and Rellier, 2003] specifies in a flexible manner the plan that organizes the activities in time, the constraints that have to be satisfied in order to make them executable, the adjustments of the plan when particular events occur, and the preferences used to select the activities to be executed. Plan revision and execution must be interleaved because the external environment changes dynamically beyond the control of the farmers (due to weather influence in particular) and because relevant aspects are revealed incrementally. The commitment to particular actions must be delayed until run-time conditions are known. In particular, what can be executed is strongly constrained by the availability of resources and state-dependent requirements on the operations suggested by the plan. This paper focuses on the computational framework that supports the representation and simulation of such plans.

Because the nature of this mental object in the mind of managers is still largely unknown, we only aspire at providing the declarative means to specify the various types of constraints bearing on the activities involved in the production process, and making such plans dynamically interpretable by a mechanism that simulates the management behavior. Due to evolving and unpredictable circumstances the plans must be flexible with respect to the constraints that express what should be done, when or under what conditions in relation to the other activities and state of the production system. The commitment to executing particular activities is delayed until run-time conditions are known.

This paper is devoted to the presentation of:
- the modeling of flexible plans as a set of activities constrained by operators that play the role of control constructs used to specify sequential and concurrency composition, iteration, optional execution, choice between alternatives and wrapping of activities;
- the algorithm in charge of the runtime interpretation of the plan seen as a program.

2.   MODELING PLANS

2. 1 Activities and primitive activities

 The basic structure in a plan is the concept of activity. In its simplest form, an activity, which is

then called a primitive activity, specifies something to be done on a particular biophysical object or location (e.g. a mob, a plant, a field or a set of these) by a performer (e.g. a worker, a robot or a set of these). Besides these three components, a primitive activity is characterized by local opening and closing conditions, defined by time windows and/or predicates referring to the biophysical state. These conditions are of use to determine at any time the activities that are eligible for execution consideration. For this purpose any activity has a status taking value in the set: *sleeping*, *waiting*, *open*, *closed* and *cancelled* (explained later).

The something-to-be-done component of a primitive activity is an intended transformation called an operation (e.g. the harvesting operation). The execution of an operation causes changes to the biophysical system, also called its effect, which are expressed in the form of values given to some variables of the biophysical systems. These changes are usually not instantaneous and take place progressively during the period of execution. In order to have the effect realized consistently with its definition the operation must satisfy some enabling conditions that refer to the current state of the biophysical system (e.g. the field to be processed should not be too muddy).

Primitive activities can be further constrained by adding temporal relations between them (sequencing, concurrency) and by using programming constructs enabling specification of choice of one activity among several, iteration, grouping and optional execution. To this end, a set of composition operators are used, the most important of which are: *before*, *meet*, *overlap*, *co-start*, *equal*, *or*, *and*, *iterate*, and *optional* (see next subsections). Any activity involving a composition operator is said to be non-primitive; a composition operator applied to an activity (primitive or not) defines another activity that may also be given local opening and closing conditions. A non-primitive activity is called the mother activity and the activities that are the arguments of the operator are called the child activities. The opening and closing of a non-primitive activity depends on its own local opening and closing conditions (if any), and of those of the underlying primitive activities that play a role through the composition operators. All the activities are connected; the only activity that does not have a mother is the plan. The plan is flexible in the sense that two different sequences of events are likely to yield two different realizations of the plan. The opening date of the same activity will not be the same in the two cases. Moreover some activities may be cancelled in one case and not in the other if they are optional or subject to context-dependent choices.

The passing of time and the evolution of the state of the production system may make true the conditions that govern the changing of status of the primitive activities. The change of status of activities is realized at particular times specified by the manager and when an operation is completed. Any change of status of an activity is propagated to the activities that are directly or indirectly connected to it via composition operators.

The meaning of the possible values of an activity status can now be explained. The value *sleeping* is given to all activities at creation time. It means that the opening and closing conditions do not have to be examined yet. The status turns to *waiting* as soon as the opening activities have to be examined. For instance, as soon as an activity finishes it becomes necessary to monitor those following it in a sequence specified with a *before* operator. The nominal plan is declared *waiting* at the starting time of a simulation. The status of an activity turns to *open* when its opening conditions are satisfied. The status changes from *open* to *closed* when the closing conditions are satisfied or, in case of primitive activity, when the underlying operation is completed. The status turns to *cancelled* when the activity becomes of no interest; this happens, for instance, once a choice among alternatives specified through the *or* operator has been made, making *cancelled* the non-selected alternatives.

The meaning of each operator used to construct à new activity by constraining other activities is defined by two sets of rules specifying:
- the preconditions that must be satisfied by the mother activity in order to enable the change of status of some of the child activity and vise versa;
- the post-conditions or effects of any change of status of one of the mother or child activities on the others.

Each type of activities constructed with such operators are visited in turn in the next subsections.

2.2 Sequencing constraints

To specify that two or more than two activities must be performed successively without any overlapping in the interval of time of their execution one can use the *before* operator and apply it to the child activities given in the order of the sequence desired. In other words, the activity *before(A B)* imposes that the activity B cannot have the status *open* before the status of A is *closed*. The order in time of the sequence is expressed by the order of the arguments of the operator. Any activity constructed using the *before* operator has two extra properties that enable specification of, if necessary, the delays between the opening of two consecutive activities, and between the closing of one of them and the opening of the next one.

The change of status of any of the involved activities is subject to the following preconditions. In order for the mother activity status to become:
- *waiting* (resp. *open*), its first child must be allowed to turn to *waiting* (resp. *open*);
- *closed*, its last child must be allowed to turn to *closed*.

In order for the first child to become:
- *waiting* (resp. *open*), the mother must be allowed to turn to *waiting* (resp. *open*).

In order for any other child than the first one to become:
- *waiting*, the preceding activity must be *closed* or allowed to turn to *closed*.

In order for last child to become:
- *closed*, the mother must be allowed to turn to *closed*.

The effect of a change of status of any (mother or child) activity follows the following rules.
As soon as the mother turns to:
- *waiting* (resp. *open*), the first child turns to *waiting* (resp. *open*);
- *closed*, the last child turns to *closed*.

As soon as a child activity turns to:
- *waiting* and if it is the first child then, the mother turns to *waiting*. Otherwise, the preceding child tuns to *closed* (if not already so);
- *open* and if it is the first child then, the mother turns to *open*;
- *closed* and if it is the last child then, the mother turns to *closed*. Otherwise, the next child turns to *waiting* if possible.

Another operator used to specify a sequence is *meet*. It is very similar to *before* except that there should be no delay between the closing of a child and the opening of the next one. The only modifications on the preconditions and effects to a change of status of any of the involved activities are the following.
In order for any child but the first one to become:
- *waiting* or *open*, the preceding activity must be *open* and allowed to turn to *closed*.

In order for any child but the last one to become:
- *closed*, the next activity must be allowed to turn to *open*.

The effect of a change of status of any (mother or child) activity follows the following rules.
As soon as a child turns to:
- *open* and if it is the first child then, the mother turns to *open*. Otherwise, the preceding activity turns to *closed*.
- *closed* and if it is the last child then, the mother turns to *closed*. Otherwise, the next child turns to *open*.

Actually a *meet* activity behaves like a *before* activity involving no delay.

2. 3 Concurrency constraints

Several operators enable to specify that some activities have to remain open concurrently for some time. For instance, using the *overlap* operator, one can express that the interval of time in which the child activities (two or more) have the status *open* must intersect and the order of the opening of the children is also the order of their closing. Therefore this operator defines a ranking over the children; this is expressed through the order of its arguments. The mother activity constructed using the *overlap* operator has three extra properties that enable specification of, if necessary, the delays between the opening of two consecutive (wrt. the order of the arguments) activities, between the opening of a child and the closing of the preceding one, and between their closing.

The rules stating the preconditions are the following.
In order for the mother to become:
- *waiting* (resp. *open* or *closed*), its first child must be allowed to turn to *waiting* (resp. *open* or *closed*).

In order for the first child to become:
- *waiting* (resp. *open*), the mother must be allowed to turn to *waiting* (resp. *open*).

In order for any child other than the first one to become:
- *waiting*, the preceding activity must be *closed* or allowed to turn to *closed*;
- *closed*, the preceding child must be *closed*.

In order for the last child to become:
- *closed*, it must be allowed to turn to *closed*.

In order for any child but the last one to become:
- *closed*, the next activity must be *open*.

The effect of a change of status of any (mother or child) activity follows the following rules.
As soon as the mother turns to:
- *waiting*, the first child turns to *waiting*;
- *open*, the first child turns to *open* and the next turns to *waiting* if possible.
- *closed*, the last child turns to *closed*.

As soon as a child turns to:
- *waiting* and if it is the first child then, the mother turns to *waiting*. Otherwise, the preceding child turns to *open* if it is *waiting*;
- *open* and if it is the first child then, the mother turns to *open*. If it is not the last child, the next one turns to *waiting* if possible;
- *closed* and if it is the last child then, the mother turns to *closed*.

Other operators enable the specification of other kinds of concurrency. For instance, the operator *inclusion* can be used to constrain the intervals of time in which the children are open to be nested (each fitting within the one immediately larger).

The operator *co-start* (resp. *co-end*) imposes the simultaneous opening (resp. closing) of the children. The operator *equal* imposes that the children be open simultaneously and closed simultaneously.

2. 4 Iteration

The operator *iterate*, which has a single argument activity, specifies that the child activity be repeated within the time in which the mother activity is *open*. The mother must be given opening and closing conditions and the child or descendant activities should not appear elsewhere in the plan. The mother constructed using the *iterate* operator has two extra properties that enable specification of, if necessary, the delays between the opening of two consecutive iteration of the child, and between the closing of the child and the opening of its next iteration. The minimum and maximum numbers of iterations may also be specified. The only preconditions to a change of status of the child are that the mother be *waiting* or *open* in order for the child to turn to *waiting*, and that the mother be *open* in order for the child to turn to *open* or *closed*.

Concerning the effects, as soon as the mother activity turns to:
- *open*, the child turns to *waiting* if possible;
- *closed*, the child turns to *open* and the next turns to *waiting* if possible.

As soon as the child turns to *closed*, it turns to *waiting* unless the closing conditions of the mother are satisfied at that time.

The iteration process, which is controlled by a specific procedure, duplicates (instantiates in fact) the child activity as needed in agreement with the constraints of delay between repetitions and of limitations of the number of iterations if provided. These copies have a status changing from *sleeping*, to *waiting*, from *waiting* to *open*, from *open* to *closed*, and, exclusively for this case, from *closed* to *waiting*. These transitions continue as long as the mother is *open*.

2. 5 Optional activity

The *optional* operator applied to an activity expresses that if this one cannot be realized (i.e. it is too late with respect to the closing interval or the closing predicate cannot be satisfied) then, it is not a sufficient circumstance to declare the plan invalid. In other words, this operator enables specification of the child activity that should be realized if possible. The child or descendant activities should not appear elsewhere in the plan if not declared optional there too. The status of the mother can change to *waiting* if the child can turn to *waiting*. Analogous preconditions hold when substituting *waiting* by *open* or by *closed* and by permuting

child and mother. The effects rules follow from the precondition rules (e.g. the child becomes *open* as soon as the mother becomes *open*).

When a mother activity made with the *optional* operator cannot be realized it status is forced to turn to *closed*.

2. 6 Disjunction and conjunction

The *or* operator enables specification of a possibility of choice between the child activities. When one of them is chosen (this can only be done after resource allocation which is not addressed in this paper) the others are turned to *cancelled* and therefore can no longer be considered for execution. The rules stating the preconditions are the following.
In order for the mother to become:
- *waiting* (resp. *open*), there must be at least one child that can be turned to *waiting* (resp. *open*);
- *closed*, all the children must be *closed* or *cancelled* or allowed to turn to *closed*.
In order for any child to become:
- *waiting* (resp. *open* or *closed*), this child must be allowed to turn to *waiting* (resp. *open* or *closed*).
The effect of a change of status follows the following rules.
As soon as the mother turns to:
- *waiting* (resp. *open*), all the children that are allowed to turn to *waiting* (resp. *open*) do so;
- *closed*, the only child that is still *open* turns to *closed*.
As soon as any child turns to:
- *waiting* (resp. *open*), the mother turns to *waiting* (resp. *open*).
As soon as the child turns to:
- *closed* (the other children being *cancelled* at this moment), the mother turns to *closed*.

The *and* operator enables to specify that the set of activities constituting the child activities should be realized so that the mother activity can end up *closed*. This operator plays the role of a wrapper. The rules stating the preconditions to a change of status are the following.
In order for the mother activity status to become:
- *waiting* (resp. *open*), there must be at least one child that can be turned to *waiting* (resp. *open*);
- *closed*, all the children must be *closed* or allowed to turn to *closed*.
In order for any child to become:
- *waiting* (resp. *open*), this child must be allowed to turn to *waiting* (resp. *open*).
The effect of a change of status follows the following rules.
As soon as the mother turns to:
- *waiting* (resp. *open*), all the children that are allowed to turn to *waiting* (resp. *open*) do so;

- *closed*, the children that are still *open* turn to *closed*.

As soon as any child turns to:
- *waiting* (resp. *open*), the mother turns to *waiting* (resp. *open*);
- *closed*, the mother turns to *closed* if all children are *closed* or can turn to *clos*ed.

3. UPDATING THE ACTIVITIES

3. 1 Algorithm

The advance of time and the evolution of the production system (the biophysical system in particular) may make true the opening and closing conditions of the activities. The updating of the status of the activities occurs at either examination times specified by the manager (typically at discontinuity points induced by new day or new week) or when an operation is terminated. The change of status is realized by a procedure that essentially checks that the opening and/or closing conditions can be satisfied and that the constraints linking this activity to others would be satisfied if the change proceeded. This procedure, applied to the plan, causes a recursive examination of all the activities that are not *sleeping*, *closed* or *cancelled*. Any activity whose change of status is validated is updated and the change is propagated immediately to the connected activities.

Normally the status updating process is repeatedly invoked until the plan is closed. In some cases, the plan cannot be closed, which reveals a plan failure. Such an inconsistency situation occurs when some preconditions to change cannot be satisfied (e.g. a *meet* activity in which the second child cannot be open although the first has just been closed). In other words, this happens when an activity that is not optional can no longer be open or when it cannot be closed without violating constraints that link them to other activities by composition operators. A more formal presentation of this updating process is given through the pseudo-code of the main procedures.

```
procedure: Update(activity)
    if      activity.situation not waiting and
            activity.situation not open
    then    return
    if      {activity.situation = waiting and
            it is no longer possible to open} or
            { activity.type = primitive and
            activity.situation = 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.situation = open and
            it is no longer possible to close
```

```
    then exit("Plan failure")
    switch activity.type
        case primitive
            if      ?OpeningValid(activity)
            then    TurnToOpen(activity)
        case iteration
            if      ?OpeningValid(activity)
            then    TurnToOpen(activity)
            if      situation = open
            then switch child.situation
                case sleeping
                    TurnToClosed(activity)
                case waiting
                    if  ?ClosingValid(activity)
                    then TurnToClosed(activity)
        case others
            for     each child do Update(child)
```

Two important predicates are used in Update: ?OpeningValid, ?ClosingValid. They return true if it is legal to open or close the argument activity. They call the two activity-dependent predicates ?CheckSonsIfOpen and ?CheckIfSonOpen. The latter, together with ?CheckSonsIfWaiting, ?CheckIfSonWaiting, ?CheckSonsIfClosed, and ?CheckIfSonClosed, implement the preconditions to changes defined for each composition operator. They themselves call ?OpeningValid, ?ClosingValid and ?WaitingValid. These three predicates are very similar in principle. The pseudo-code of ?OpeningValid is given below. For clarity, this code does not include all the bookkeeping structures and tests necessary to avoid loops.

```
predicate: ?OpeningValid(activity)
  if activity.situation = open  then  return  true
  else
     if  {activity.situation = 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
        else  return false
     else return false
```

Note that the predicates ?OpeningValid, ?ClosingValid and ?WaitingValid are also used in the operator-dependent procedures that implement the effect of a change of status of an activity.

Update calls the procedures TurnToOpen and TurnToClosed. Together with TurnToWaiting each of these procedures realizes the due changes of status of the argument activity and propagates the effect to the connected activities. Once they are called (either by Update or at the beginning of the simulation when the plan status is forced to change from *sleeping* to *waiting*) they perform all the required changes in the plan according to the operator-dependent rules.

3. 2 Example

An application of the concepts and mechanisms defined in the above sections has been used to describe glasshouse production system for tomatoes by Jeannequin et al. [2003]. For illustration, we consider here a highly simplified management plan that is actually only a part of a real one in this domain; this part should normally be considered with the other parts at the same time because they are likely to interact. The plan is the following:

before(iterate(PRUNING1), iterate(optional(PRUNING2)))

It expresses that two series of pruning activities have to be done successively and the pruning activities in the second series are optional. Both PRUNING1 and PRUNING2 are primitive activities that consist in applying a Prune operation to the plants of a particular glasshouse compartment. This operation removes young fruits from the most recent truss so as to leave only a limited number of them and prevent small sized fruit. The above two activities differ only by the resources that they require: the first one needs one worker of a particular type (e.g. highly qualified) whereas the second one needs one too but of another type (e.g. temporal labor). We assume that w1and w2 are workers of the first and second type respectively. w1 is available from day 0 to day 30 whereas w2, is hired from day 30 to the end of the season and might nevertheless be unavailable from time to time at random due to other duties. We assume he might be off for 6 consecutive days every 2 weeks (15 days) but he must stay at least five days when he comes back to his glasshouse job. The area of the glasshouse compartment is equal to 10 units and the pruning speed of a worker is 2 units per day.

The temporal specifications in the various activities are expressed on a daily scale. It is assumed that the plan itself (i.e. the before activity) has opening and closing windows equal to [0, 60] and [60,60] respectively. The opening window of the first pruning activity in the first series is [0, 5]. When a pruning activity is open at time t the opening window of the potential next iteration in the series is set to [t+10, t+15]. Any pruning activity has a closing predicate that forbids its closing later than 10 days after the execution of the underlying operation has started. The two arguments of the before activity have [0, ∞] as opening windows; their closing window are [30, 60] and [60, ∞] respectively. Finally the before activity is specified that the opening and closing windows of the potential first iteration of the second series is set to [t+10, t+15] where t is the opening date of the last iteration in the first series. Since the availability of w2 is stochastic the outcome of running the plan is stochastic too. One of the possible realisations is considered next.

The first series involves three pruning activities that are opened as soon as possible with respect to the delay constraints (at days 0, 10 and 20 respectively). They are never interrupted by resource unavailability so the execution of the operation always extend over 5 consecutive days. The first pruning activity in the second series behaves similarly for the same reason. At day 40 another pruning activity is opened but the operation cannot be performed because worker w2 is not available. Since w2 comes back only at day 46 and a prune operation cannot start executing later than 15 days after the opening of the pruning activity, this optional activity cannot be performed and is simply closed. The following candidate activity is opened at day 50 (i.e. 10 days after the previous opening). The prune operation is executed at days 50 and 51 when w2 is available. This is not enough to complete the activity, which resumes as soon as w2 is back at day 58. The operation ends at day 60, which complies with the delay requirement that the activity ends within 10 days after its beginning. As specified, the execution of the plan stops at the end of day 60.

4. CONCLUSION

The conceptual and computational model of plans presented in this paper has been developed for and inspired by production management problems in agriculture. It seems nonetheless relevant in any production process that involves a single manager and that highly depends on uncontrollable factors, thus requiring flexible management plans.

Besides the runtime interpretation of plan other important tasks such as resource allocation or strategy adjustment are involved in plan execution and have not been addressed in this paper although already implemented.

Future research effort will be devoted to the issue of preference processing including consideration of subsidiary goals and anticipation of likely future.

5. REFERENCES

Jeannequin B., R. Martin-Clouaire, M. Navarrete, and J.-P. Rellier, Modeling management strategies for greenhouse tomato production. CIOSTA-CIGRV Congress, Turin, Italy, Sept. 22-24, 506-513. 2003.

Martin-Clouaire R., and J.-P. Rellier, A conceptualization of farm management strategies. Proc. of EFITA-03 conference, Debrecen, Hungary, July 5-9, 719-726. 2003.