REPRESENTING AND INTERPRETING FLEXIBLE PRODUCTION MANAGEMENT PLANS

Roger Martin-Clouaire, Jean-Pierre Rellier
INRA, Unité de Biométrie et d'Intelligence Artificielle
BP52627 Auzeville, 31326 Castanet Tolosan, France.
rmc@toulouse.inra.fr, rellier@toulouse.inra.fr

ABSTRACT

In agriculture the production management difficulties stem from the high influence that uncontrollable factors like weather or pests have on the biological 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 study of the underlying production management behavior.

1. INTRODUCTION

This paper presents a plan representation language designed for production management tasks that are highly dependent on exogenous uncontrollable factors. The plan representation approach is motivated by applications to the study of agricultural production processes such as dairy or crop production systems. The yield, quality and costs of agricultural productions are inherently affected by weather, diseases, pests and other factors that are highly uncertain. A farm's profitability and overall financial health is, of course, highly dependent on the mechanisms in place for performing timely and efficiently the required agronomic operations and for mitigating the risk exposure.

The biological nature of agricultural production makes it fundamentally different from manufacturing. Uncertainty in manufacturing concerns mainly the production goals (the demand) and to a lesser extent the availability of resources (e.g. machine breakdown). In agriculture, uncertainty affects the determinism of the actions and forces context-dependent courses of actions to be adopted to cope with threat or exploit opportunities. The production processes in manufacturing is fully manmade and can be designed in a way supporting the planning and scheduling of operations. In agriculture the production process goes on even if no action is performed; external natural inputs (light, energy) constitute the primary driving factor. In addition, the performance criteria are of different types such as minimizing the timespan in manufacturing versus keeping the production risk under control in agriculture.

Nevertheless, despite the pervasiveness of uncertainty and variability in farm production processes, the decision making behavior is far from being purely reactive. Indeed the production processes offer a great deal of structures and regularities from one year to the other, which enable farmers to plan roughly the intended activities required by the overall production objective.

Capturing how the flexible temporal organization of activities can be specified and how such a specification can be used for on-line determination of what actions are licit for execution is the subject of this paper. The underlying objective behind this modeling endeavor is to be able to deal with agricultural production management behavior as an object of scientific consideration and to conduct virtual experimentation on it by computer simulation. Explicit representation and simulation are also a means to ease communication, learning and design of possible management behaviors.

The next section provides some background about production management in agriculture. The plan representation language is presented in Section 3. The algorithm that maintains the status of the plan activities is given in Section 4. Section 5 points out some related works.

2. PRODUCTION SYSTEM ARCHITECTURE

As shown in Figure 1, an agricultural production system is seen as an entity situated in what is called the external environment (e.g. the climatic and economic context) and can be decomposed into three interactive subsystems: the manager, the operating system and the biophysical system. A production system is an active entity in the sense that it is the repository of processes and has inputs (physical or informational), outputs and an event agenda. The processes are controlled by events (straight lines) of the agenda.

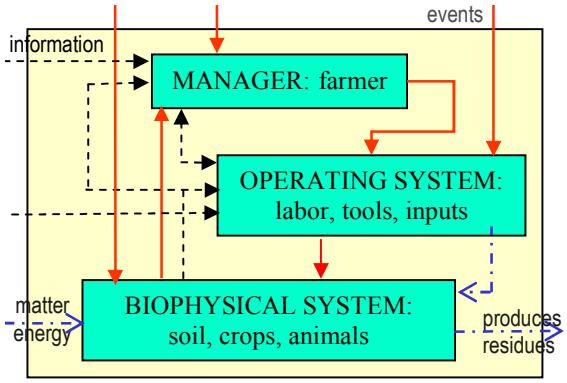

Figure 1. Agricultural production system

The biophysical system is composed of biophysical entities (e.g. crop, animals) that usually have their own processes (e.g. photosynthesis, physiological functions). Among the events controlling these processes are those triggered by the execution of the operations performed by the operating system. The inputs are material inputs (e.g. fertilizers provided by the operating system) and energy either coming from the external environment or provided by the operating system. The processes may generate particular events triggered by significant changes of the biophysical system state. Thus the biophysical system may also include some sensing and alarm devices.

The manager stands for the farmer having the responsibility of achieving the overall production system objective. In our model, the manager holds a management strategy that drives the behaviors of the operating system and, indirectly, of the biophysical system. A strategy is a handcrafted construct that specifies a kind of flexible nominal plan coming with its context-responsive adaptations and the necessary implementation details to constrain the stepwise determination and execution of the actions to perform.

Since the production process is highly influenced by factors beyond his control the farmer must pay special attention to the robustness of his strategy so as to work acceptably well in almost all climatic scenarios and be responsive to important contingencies whose effects can be eliminated or mitigated by proper agronomic practices. Therefore agricultural production management must rely on a decision making behavior that is both plan-based and reactive.

With accumulation of experience and advice, farmers have learned to design their own temporal organization of farming activities. The planning is done consistently with overall objective, resource limitation and intended tactic, with their own perception and understanding of the production system characteristics and with particular events that have to be monitored and reacted to.

The manager's processes are responsible for:

- monitoring the occurrence of new events and scrutinizing salient aspects of the current state of the production system (mainly in the biophysical system);
- updating the status of the activities in the nominal plan depending on changes of the system state and the passing of time (e.g. some activities may be obsolete while other may become ready for execution consideration);
- revising the management strategy in situations recognized beforehand to require such adaptations;
- generating the sets of activities that are feasible (i.e. consistent with the nominal plan and thus open to further consideration for execution) and providing the necessary implementation details that constrain the dynamic allocation of resources.

Every time the manager is activated the result of his work (advocated sets of activities and requirements) is handed over to the operating system that has to execute them using the resources it is equipped with (e.g. labor, tools). The operating system utilizes, within its autonomy, its own problem solving procedure to derive the selected set of executable activities. It has processes, in particular, to:

- allocate resources to the activities;
- select the preferred set of activities in case of concurrency.

The execution of the current set of operations continues until a change on the resources occurs (end of an operation or end of working hours). Such an event may be followed by a new scheduling of activities to execute or by a transfer of control to the manager or by nothing if the plan is exhausted.

The next section focuses on the representation of plans.

3. REPRESENTING PLANS

3.1 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. These changes take place over a period of time. An operation affects either individual objects in a collection of processed objects (e.g. plants in a greenhouse population) or objects having numerical characteristics (e.g. area). The speed is defined as a quantity (e.g. number of items, area) processable in a unit of time. The duration of the operation is the ratio of the total quantity by the speed. 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 and by using programming constructs enabling specification of temporal ordering, iteration, grouping and optional execution. To this end, a set of composition operators are used such as *before*, *iterate*, and *optional* that are presented in the next subsections. Other operators are utilized to specify choice of one activity among several (*or*), grouping of activities (*and*) and concurrency among some of them (e.g. *overlap*, *co-start*).

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 a 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 vice versa;
- the post-conditions or effects of any change of status of one of the mother or child activities on the others.

The cases of the before, iterate, and *optional* operators are visited in turn in the next subsections.

3.2 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. 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 turns 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.

3.3 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 time windows, or opening and closing predicates, or the maximum and minimum number of replication or any combination of the above possibilities. 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 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*.

As soon as the mother activity turns to:
- *open*, the child turns to *waiting* if possible;
- *closed*, the child turns to *closed* if possible.

As soon as the child turns to *closed*, it is set immediately to *waiting* unless the mother's closing conditions 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*.

3.4 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 effect 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 its status is forced to turn to *closed*.

4. UPDATING THE ACTIVITIES

4.1 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 *waiting* or *open*. Any activity whose change of status is validated is updated and the change is propagated immediately to the connected activities.

A more formal presentation of this updating process is given through the pseudo-code of the Update procedure that follows.

```
procedure: Update(activity)
    if      activity.situation not waiting and
            activity.situation not open
    then    return
/* beginning of plan failure detection */
    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")
/* end of plan failure detection */
    switch activity.type
            case primitive
                 if      ?OpeningValid(activity)
                 then    TurnToOpen(activity)
                 if      ?ClosingValid(activity)
                 then    TurnToClosed(activity)
                /* ?ClosingValid includes test relative to minimal
                percentage of execution realization required, typically
                100%*/
            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)
```

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 should be 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.

Two important predicates are used in Update: ?OpeningValid, ?ClosingValid. They return true if it is legal to open or close the argument activity. ?OpeningValid calls the two activity-dependent predicates ?CheckSonsIfOpen and ?CheckIfSonOpen. The latter two, together with the predicates ?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}; return true
        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.

4.2 Example

The concepts and mechanisms defined in the above sections have been used to describe glasshouse production system for tomatoes (Jeannequin *et al.*, 2003)]. To illustrate this application, we consider here an extremely simplified management plan that is actually only a small 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 through their resource demands. 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 worker too but of another type (e.g. temporal labor). We assume that w1 and 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 windows are [30, 60] and [60, ∞] respectively. Finally the *before* activity is specified such 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 realizations is considered next and shown in Figure 2.

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 extends 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 previous 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.

5. RELATED WORKS

Several agent behavior specification approaches have been published in the AI robotic literature in recent years. Logic-based agent languages such as those of the Golog/ConGolog family (De Giacomo *et al*., 2000) were designed primarily to support formal reasoning about current and potential agent activities to ensure that some properties are met. They rely on explicit symbolic representation of the environment of the agent and on action theories that enable to express relationships between fluents (properties that change from a situation to another), the effect of actions on these fluents and reason about them. ConGolog allows specification of complex plans that are kinds of control procedures. It use programming constructs such as sequencing, parallel execution, conditional statements, non-deterministic choice of actions, iteration, procedure call. A ConGolog program when executed uses an extended version of situation calculus to simulate the changes in the world so as to decide on the executability of an action before actually executing it and also to figure out which branch to take faced with a conditional statement. The off-line interpreter verifies the executability conditions, evaluates the conditions in the conditional statements and makes choice at non-deterministic choice points before actually executing the program. Therefore, the correctness of a ConGolog program with respect to a goal or a particular property can be verified off-line. The sequence of actions generated as a trace of the verification proof can then be executed.

The main difference with our approach is that our interpreter can only determine repeatedly the actions that are legible to execution; the non-executability is a property that is eventually revealed when a dead end is met with. Actually, for the target application domain, we are more interested in a probabilistic assessment of the non-executability; a plan that does not work in very extreme climatic scenarios (e.g. severe drought) may not necessarily be rejected in agriculture. A situation of non-executability of the plan uncovered by simulation calls for a modification of the plan or of the conditional adjustments that should be used to make an modification of the plan in

reaction to the occurrence of an anticipatory event. In addition, we address management problems that involve rich temporal and procedural constraints on and between the activities. We have paid special attention to the plan intelligibility provided by the language. The actions have complex and highly uncertain consequences that can hardly be incorporated in an action theory enabling to reason about their anticipated effects. It is still an open question whether formal approaches like ConGolog could scale up to problems of this size, involving continuous changes (see (Martin, 2003) however).

Reactive plan frameworks (see SPARK (Morley and Myers, 2004) for one of the latest, a member of the PRS family (Ingrand *et al*., 1992)) are also related to the present work in the sense that they provide languages to express procedural organizations of actions. They are equipped with an execution mechanism capable of implementing open-ended responsive decision making behavior based on high level control constructs. These languages do not offer however rich ready-to-use primitives to express temporal constraints on the activities. Indeed a PRS agent will have a set of pre-defined procedures (called plans or knowledge areas) that specify how to achieve goals or perform activities. Each plan includes a trigger, which indicates when it should be considered for use, a context, which indicates when it is valid for use, and a set of actions, which specify sequentially how to achieve goals or react to events. The temporal reasoning capabilities are rather limited and the ability to maintain a sense of continuity in the application of a nominal plan is hard to reproduce.

The kind of flexible temporal constraints that is used in our plan representation framework is also present in the COMIREM system (Smith *et al*., 2005) that promotes an opportunistic interactive planning paradigm. In this system resource allocation decisions are made incrementally as availability constraints and activities coming from the plan become known.

6. CONCLUDING REMARKS

We have presented a special purpose plan representation language designed for production management tasks that are highly dependent on exogenous uncontrollable factors and that involves activities constrained by rich temporal properties. The representation framework (Martin-Clouaire and Rellier, 2004) and the discrete event simulation engine (Rellier, 2005) that runs the various processes are implemented as a C++ package.

As pointed out in the previous section, the problem of designing purposeful programmable action behaviors in open environments is also addressed by the planning/scheduling and autonomous agent communities in artificial intelligence. In these approaches the emphasis is more on automatic construction of plans and formal verification of plan properties or on execution performance. Because we only aim at simulating decision behavior we give greater importance to the design of a rich representation language that can incorporate the kind of knowledge used by production managers in practice. The language must grant enough flexibility, enabling to avoid too early decisional commitment and to interleave procedural reasoning and resource allocation at execution time.

As pointed out in Section 2, coping with uncertainty in agricultural production management requires to perform reactive plan revisions when particular events occur. The way to adapt the plan in such cases has not been addressed in this paper. Another important aspect of our modeling and simulation undertaking concerns resources and their allocation process. This is the subject of a paper to come.

At this stage, the decision making behavior that is associated to our representation and interpretation framework works without an explicit representation of goals. This becomes necessary to take into account anticipatory decision making capabilities that are invoked in some agricultural management problems such as those taking place in infinite temporal horizon in particular. Consequently we are currently extending the framework towards a Belief-Desire-Intention (BDI) type of decision making architecture (Rao and Georgeff, 1995) in which beliefs express the decision maker current state of knowledge about the production system, intentions are the activities structured in a plan and desires are specifications about target states of the production system.

7. REFERENCES

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

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

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

Martin, Y. 2003. “The concurrent continuous FLUX”. Proc of IJCAI, Acapulco.

Martin-Clouaire, R., J.-P. Rellier. 2004.“Fondements ontologiques des systèmes pilotés”. Internal report UBIA-INRA, Toulouse-Auzeville.

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

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

Rellier, J.-P. 2005. "DIESE : un outil de modélisation et de simulation de systèmes d'intérêt agronomique". Internal report UBIA-INRA, Toulouse-Auzeville.

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

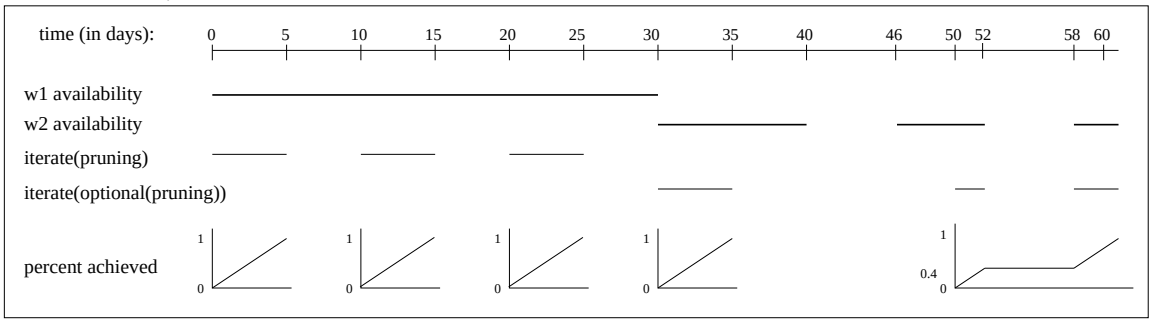

Figure 2. Interpretation of the plan given the constraints on resource availability