# Modelling and simulating work practices in agriculture

Roger Martin-Clouaire, Jean-Pierre Rellier

# Modelling and simulating work practices in agriculture

Roger Martin-Clouaire* and Jean-Pierre Rellier

INRA (Institut National de la Recherche Agronomique),
UR875 Biométrie et Intelligence Artificielle,
F-31326 Castanet-Tolosan, France
E-mail: rmc@toulouse.inra.fr
E-mail: rellier@toulouse.inra.fr
*Corresponding author

**Abstract:** Research has shown that the managerial capacities and work practices of farmers play a major role in explaining differences in economic and environmental performances. This paper presents a computer simulation framework that enables work organisation issues in agricultural production systems to be studied. This framework relies on a purposive frame-based ontology of such production systems. The paper focuses on a subpart of the ontology that concerns production activities, flexible plans and material resources. The paper also outlines the interpretation algorithms that operate on instances of these ontology concepts in any production system model constructed in compliance with the ontology.

**Keywords:** simulation; scheduling; agricultural production system; activity; plan; resource.

**Reference** to this paper should be made as follows: Martin-Clouaire, R. and Rellier, J-P. (2009) 'Modelling and simulating work practices in agriculture', *Int. J. Metadata, Semantics and Ontologies*, Vol. 4, Nos. 1/2, pp.42–53.

**Biographical notes:** Roger Martin-Clouaire holds a Masters in Biomedical Engineering (1982) from Saskatchewan University (Canada) and a PhD (1986) in Artificial Intelligence (AI) from Toulouse University. He joined Institut National de la Recherche Agronomique (INRA) in 1987 as a research scientist. His main research area concerns the modelling and simulation of agricultural production systems and, in particular, decision-making processes involved in production management. He is currently director of the laboratory Unité de Biométrie et Intelligence Artificielle.

Jean-Pierre Rellier joined INRA in 1976. In his early career, he worked as a statistical analyst in the area of cropping systems. In the mid-1980s, he became a member of the national group on agricultural expert system development. Since 1988, he has been a software engineer in the Unité de Biométrie et Intelligence Artificielle, where his main areas of interest concern the methodological aspects of complex system modelling and simulation.

## 1 Introduction

### 1.1 *Work practices in agriculture as an object of scientific investigation*

Farming involves the input of resouaarces (seed, fertiliser, pesticides, time, labour, etc.) to natural systems driven towards the harvesting of outputs for sale (biomass, grains, livestock, etc.). The complex interaction between natural and human-controlled processes is at the very heart of agricultural production. As a production manager, the farmer makes decisions about the timing, combination and implementation of technical operations (tilling, planting, fertilising, irrigating, spraying, harvesting, feeding livestock, etc.) with the aim of achieving his objectives. The farming business is risky because operation outputs are subject to both unpredictable natural events (weather, disease, etc.) and changing economic factors (market demand, price fluctuation, etc.).

Tough competition combined with a concern for environmentally acceptable practices and a desire for better working conditions make farm production management a complex task, resulting in a greater demand for farm management research. Farming practices are becoming an increasingly prominent issue in policy development and market positioning. Consequently, previously acceptable farming practices must be reassessed and economically viable alternatives sought. Clearly, the important aspects of production management regarding risk control, changes (new practices, products and techniques) and more stringent resource allocation require innovative approaches that recognise and focus on the holistic, dynamic and human dimension of farm systems.

### *1.2 A simulation approach*

Work practices are ways of structuring things to be done or ways in which things are done. Studying management and work practices implies a strong emphasis on identifying which activities are relevant for a given production objective, how they are interdependent, what the preconditions to their execution are, and how they should be structured in time and space to meet any constraints and achieve the desired outcome.

Most farm managers develop a functional understanding of the work they do, but the scope and complexity of their work practices often make them difficult to comprehend fully. The biophysical processes at the core of their business are often only partially known and depend on climatic factors that are highly uncertain both during a given year and from one year to the next. The farmer's inadequacy becomes most apparent when the way the work is done must be changed, as and when new constraints (e.g., environmental regulation, market demand) are imposed.

By its ability to support virtual experimentation with a dynamic system, computer-based simulation is a very appealing approach to explore production management issues. Many simulation tools (e.g., Carberry et al., 2002) have been built to study isolated agronomic and technological aspects of the production processes, e.g., crop or livestock responses to particular farming operations. Surprisingly, little attention has been paid to the modelling and simulation of farmers' management and work practices. Studying and supporting the development of work practices by means of computer tools has as yet rarely been addressed directly or systematically as an issue in its own right, probably because modelling human decision processes is still a scientific challenge and agricultural research is more inclined towards applying the scientific knowledge to the design of material technologies (seeds, fertilisers, herbicides and machinery) than to studying the processes of on-farm decision-making. Our research work aims at providing a simulation framework for virtual experimentation enabling farming system researchers to study how management decisions are made in uncertain conditions, how activities are coordinated, how scarce resources (e.g., labour, machinery) are allocated, and how planned activities are actually implemented in situ. Such a simulation tool can be of great help to gain a better understanding of the functioning of production systems, to improve them, develop new ones and support learning processes. The originality of our simulation framework lies mainly in the provision of a representation of the farmer's behaviour as a cognitive agent interacting with and operating on a biophysical system.

### *1.3 Ontology for work practices*

Ontology (Chandrasekaran et al., 1999) is a term originally coined by philosophers to refer to the study of being or what exists. In computer science, ontology has been adopted by the Artificial Intelligence (AI) community as a means to provide a formal definition of a body of knowledge relevant for a specific purpose. Making ontology is concerned with identifying and describing the essential concepts and constraints of a domain with the help of a representation language that is based on a small set of basic meta-concepts. However, building ontology means different things to different practitioners, ranging from simple lexicons, to categorically organised thesauri, to taxonomies where terms are related hierarchically and have distinguishing properties embedded in a logical theory. Ontologies also differ in their scope and purpose. The most prominent ontologies, especially those on the web built with OWL (Smith et al., 2004), rely on frame-based representation languages equipped with powerful reasoning facilities. The formal semantics founding such ontologies enable, for instance, testing the consistency of ontology after an updating or merging operation, or inferring properties that are not literally present in the ontology.

The ontology of agricultural production systems (Martin-Clouaire and Rellier, 2006) presented in this paper does not require such inferential capabilities because the purpose is not related to ontological reasoning but rather to supporting the development of simulation models of such systems. Our ontology is part of a computer simulation framework and provides at conceptual level the means to describe farm system aspects that are relevant for studying work practices. As such, the ontology serves as a metamodel that enables the reuse of pre-formalised concepts and templates to be particularised, instantiated and then mapped into an executable dynamic model of a specific system. This ontology results from discussions with farming system specialists, our own modelling experience and review of the literature on dynamic systems, planning, workflow management (WFMC, 1996) and business process modelling.

The paper focuses on the part of the ontology that concerns the conceptualisation of technical production activities, their organisation in flexible plans and the material resources required by the activities together with the various restrictions on their availability and use. The paper also outlines the processes that operate on instances of these structural components in any production system model constructed in compliance with the ontology.

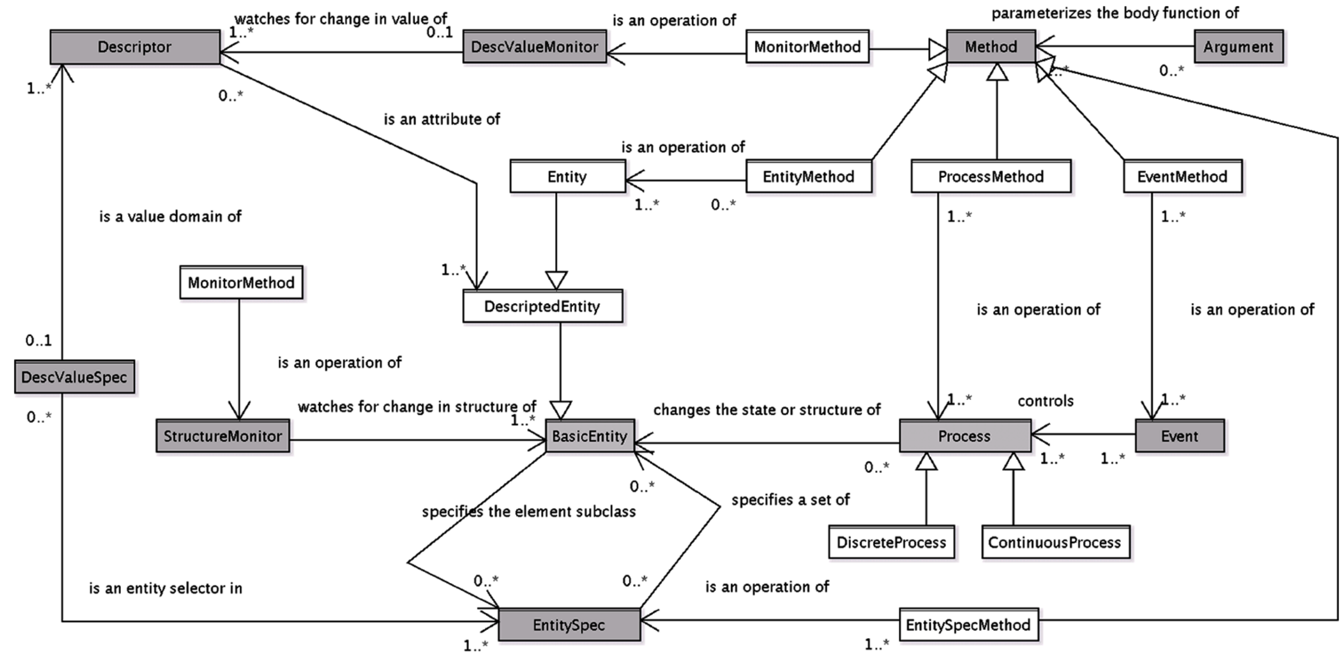
In Section 2, we sketch out the frame representation and dynamic system primitives that underlie the ontology we have developed. An informal conceptual model of an agricultural production system is given in Section 3. Deeper insight into the modelling of activities and their organisation in flexible plans is provided in Section 4. Resources and constraints on their usage are addressed in Section 5. Section 6 illustrates the use of ontology in the study of grassland-based dairy systems. Section 7 reviews related studies. Finally, in the concluding section, we summarise the contribution of this paper and outline future developments.

## 2 Ontology foundations

A frame representation (Chaudhri et al., 1998; Smith et al., 2004) has been used to formalise the corpus of domain knowledge relevant to production systems and their study by simulation. A frame is a data structure that represents a set of things, a concept or an abstraction. A frame has slots that describe the attributes or properties of the things represented by the frame. A slot can be filled by values of various types such as numbers, strings, lists, frames or procedural fragments. Each slot can be associated with value restrictions (facets) and procedures that specify reactions when a value is changed or accessed. Particular slots enable modellers to express that some frames are composed of other frames. Others allow the assertion of a frame taxonomy. This hierarchy can then be used for inheritance of slots, allowing a sparse representation. As well as frames representing concepts, a frame-based representation may also contain instance frames representing particular realisations.

The notion of frames emphasises their role for the representation of knowledge. Object-oriented programming is a programming paradigm that emphasises the role of objects as being the primary concern in the programming task; objects are represented by classes encapsulated with attributes and services defined by functions. The two concepts are often confused because they operate with overlapping terminology. Some ontology-design ideas originated from the literature on object-oriented design and the UML language (Booch et al., 2005). However, ontology development is different from designing classes in object-oriented programming. In object-oriented programming, a programmer makes design decisions based on the operational properties of a class, whereas ontology designer makes these decisions based on the structural properties of a frame. However, objects and frames are related by implementation. In our representation framework, frames are implemented by classes. The graphical notations of UML are used (see for instance, Figure 1) to communicate in a standard way the structural aspects of the ontology.

**Figure 1** UML class diagram of dynamic system foundations of the ontology




It is certainly unusual in the ontological engineering realm to let slot values be procedural codes because it is virtually in contradiction with the emphasis on enabling logical reasoning about the ontology content. Since our ontology aims rather at supporting the design and development of dynamic system models, we need to provide the means to describe behaviour. Using procedural slot values is a convenient way used to express at semantic level how things change in response to a stimulus.

Building on the base of this frame representation, we have developed three fundamental concepts for the modelling of dynamic systems: entity, process and event. These represent the structural, functional and dynamic aspects of a system, respectively (Rellier, 2005). An entity describes a kind of material or abstract item in the area of interest. The state of a system at a given moment in time is the value of the slots of the entities it comprises. A process is a specification of the behaviour of a system, i.e., of the entities composing it. Typically, the process code specifying this behaviour includes the use of methods attached to entities affected by the process. A process causes a change in state when a particular event occurs. Thus, events convey the temporality of process triggers.

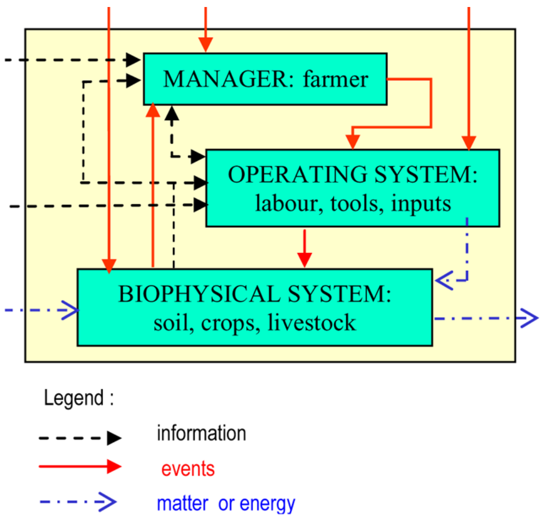
Other useful concepts have also been introduced. An entity set specification is the functional definition of a set of entities, such that the resultant set content (that may happen to be a singleton) depends on the current state of the system. Monitors are devices that simulate the mechanistic or natural reactions of entities to stimuli. They watch for changes in structure or value and trigger a procedure that implements the desired reaction. A descriptor is the encapsulation of everything that is known about a descriptive attribute (semantics, value domain, default value, current value, monitor attached). A method is the encapsulation of everything that is known about a functional attribute (semantics, returned type, parameters, code to execute). Specialised subframes of methods are designed to be attached to entities, processes, events, etc.

Actually the production system ontology consists of a set of particularisations of these concepts as shown in the next sections. The ontological representation framework, depicted in Figure 1 under the form of a UML class diagram, is implemented as a C++ package called DIESE that also includes a discrete event simulation package designed to operate on the data structures underlying the ontology. The simulation engine of DIESE carries the inferential mechanisms in charge of processing the event agenda and producing the dynamic behaviour of the system model.

## 3 Architecture of a production system

An agricultural production system (see Figure 2) is conceptually an entity situated in and influenced by what is called the external environment (e.g., the climatic and economic context). It can be divided into three interactive subsystems: the manager, the operating system and the biophysical system. A production system and the three composing subsystems are active entities in the sense that they are the repository of processes and have inputs (physical or informational), outputs and an agenda of events. The processes are controlled by the events (straight lines) of the agenda.

**Figure 2** Agricultural production system (see online version for colours)




The biophysical system is composed of biophysical entities (e.g., crops, livestock). It has processes such as photosynthesis or animal intake that specify how the biophysical entities change. 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., fertilisers 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 connected to significant changes in the state of the biophysical system. Thus, the biophysical system may also include sensors and alarm devices, modelled as monitors.

The manager is the farmer who has the responsibility of achieving the overall production system objective. In our model, the manager has a management strategy that drives the behaviours of the operating system and, indirectly, of the biophysical system. A strategy is a handcrafted construct that specifies a kind of flexible nominal plan complete with context-responsive adaptations and the relevant implementation details for the step-by-step control and execution of the actions to be performed.

Since the production process is greatly influenced by factors beyond his control, the farmer must pay special attention to the robustness of his strategy so as to work reasonably well in almost all climatic scenarios and to be responsive to important contingencies whose effects can, in most cases, be eliminated or mitigated by proper agronomic practices. Agricultural production management must therefore rely on decision-making behaviour that is both plan-based and reactive.

As farmers have accumulated experience and advice, they have learned to develop their own temporal organisation of farming activities consistently with the overall objective and resource limitations. The resulting management strategy reflects the farmer's personal work practices, which can be seen in his monitoring and observation behaviour, in his understanding of the way the production system functions, and in his appreciation of what events are important and how they should be reacted to.

The manager's processes are responsible for:

- monitoring the occurrence of new events and scrutinising salient aspects of the current state of the production system (mainly in the biophysical system)
- revising the management strategy in situations recognised beforehand to necessitate such adaptations
- updating the status of the activities in the nominal plan according to changes in the state of the system and the passing of time (e.g., some activities may be obsolete while others may now be considered for execution)
- 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 controlling the dynamic allocation of resources.

Every time the manager acts, the results of his work (advocated sets of activities and requirements) are handed over to the operating system that has to execute them or some of them using the resources available (e.g., labour, tools). The operating system utilises its own problem-solving procedure to derive the selected set of executable activities. Typically, the role of decision-maker involved in the operating system is also played by the farmer; on large farms, however, the manager might delegate this role to another person. Two essential processes are involved in the decision-making of the operating system. They aim at:

- allocating resources to activities
- applying the manager's rules of preference to select the preferred set of activities if there are concurrent options.

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

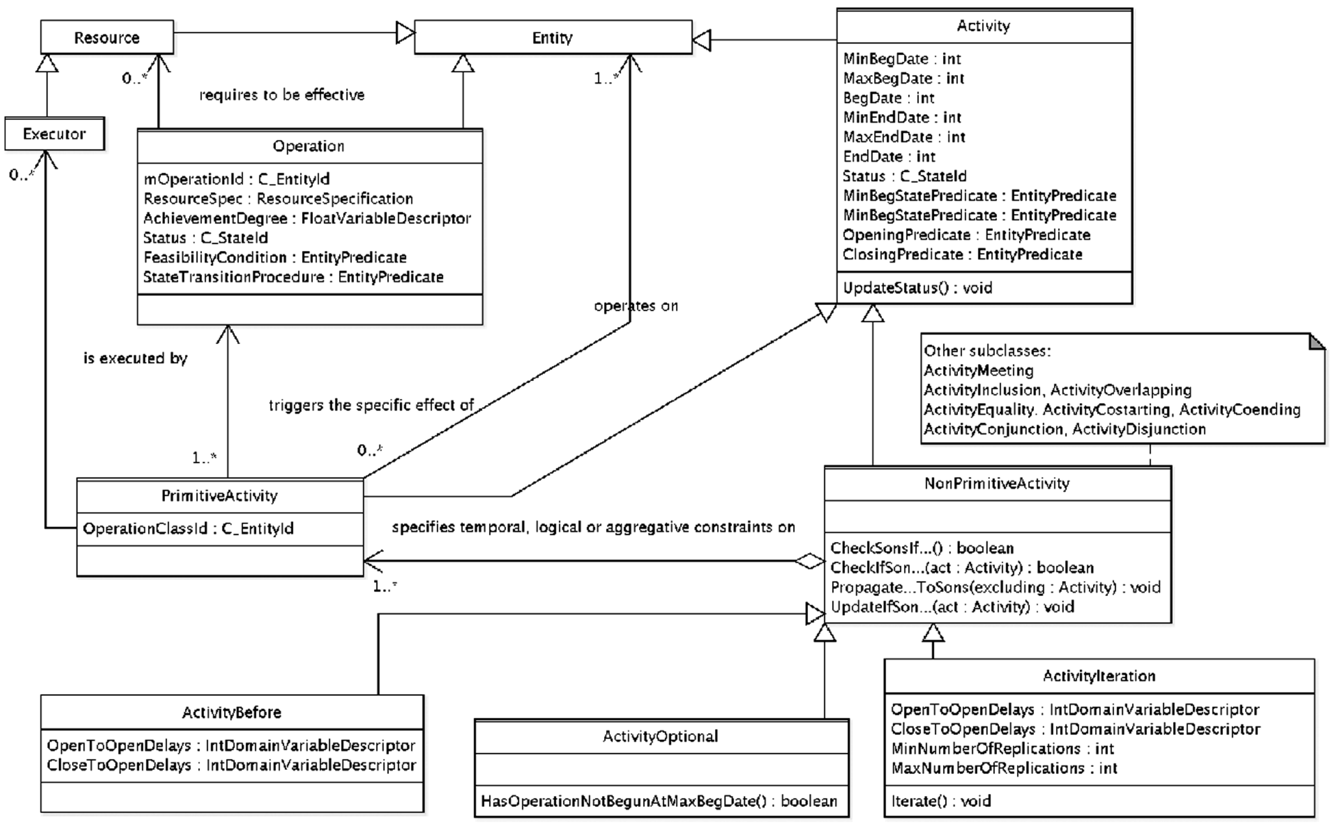
The next two sections focus on the ontological constructs developed for a rigorous representation and processing of plans and resources, respectively.

## 4 Plans and their unfolding through time

### *4.1 Primitive and composed activities*

Our ontology abides by an activity-centred conceptualisation of work organisation. In its simplest form, an activity, which is then called a primitive activity, denotes something to be done to a particular biophysical object or location (e.g., a mob, a plant, a field or a set of these) by an executor (e.g., a worker, a robot or a set of these). Besides these three components, a primitive activity is characterised by local opening and closing conditions, defined by time windows or predicates (Boolean functions) referring to the biophysical state. These conditions are used to determine, at any time, which activities are eligible for execution. For this purpose, any activity has a status within this set: *sleeping*, *waiting*, *open*, *closed* and *cancelled* (explained later). Formally, in the frame-based representation language introduced in Section 2, the notion of activity is represented as a particularised entity (see Figure 3). The slots defined for this notion are inherited by the subclasses. For instance, the non-primitive activities that are particular activities have all the properties defined at the more abstract level. As subclasses they have extra slots.

**Figure 3** UML class diagram of entities linked to 'activity'




The "something-to-be-done" component of a primitive activity is an intentional transformation called an operation (e.g., the harvesting operation). The notion of operation is also represented as a particularised entity (see Figure 3). The step-by-step changes to the biophysical system as the operation is carried out are specified in a particular functional slot (method) of the operation. These changes take place over a period of time by means of a process that increases the degree of achievement at each step of the operation until it is completed. An operation is said to be instantaneous if its degree of achievement goes from 0 to 1 in a single step. An operation affects a collection of objects resulting from the expansion of the specification of an entity set. Objects on which an operation is carried out can be individual objects (e.g., a field or a plant) or objects having numerical descriptors (e.g., an area). Speed is defined as a quantity (e.g., number of items, area) that can be processed in a unit of time. The duration of the operation is the ratio of the total quantity to the speed. To have the effect realised the operation must satisfy certain enabling conditions that refer to the current state of the biophysical system (e.g., the field to be processed should not be too muddy).

Activities can be further constrained by adding temporal relations between them and by using programming constructs enabling specification of temporal ordering, iteration, aggregation and optional execution. To this end, we use a set of non-primitive or aggregated activities having evocative names such as *before*, *iterate*, and *optional* that are presented in the next subsections. Others are utilised to specify choice of one activity among several (*or*), grouping of activities (*and*) and concurrence among some of them (e.g., *co-start*, *equal*, *include*, *overlap*). Formally, a non-primitive activity is a particularised activity. As such it might also be given opening and closing conditions. It has a relational property that points to the set of other activities directly involved in it (or constrained by it). In addition, it is equipped with a set of procedural slots that are receptacle of the semantics of the change in status specific to each non-primitive activity.

A non-primitive activity is called the mother activity and the activities that are constrained by it are called the child activities. The opening and closing of a non-primitive activity depend on its own local opening and closing conditions (if any) and on those of the underlying activities. All the activities are connected; the only one 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 realisations of the plan due to the functional nature of the specifications of activities and operations.

The passing of time and the evolution of the production system may render true the conditions that govern the change in status of the primitive activities. The change in status of activities occurs at particular times specified by the manager and also when an operation is completed. Any change in status of an activity is propagated to the activities that are directly or indirectly connected to it via the constraints enforced by non-primitive activities.

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

The principle that governs the change in status of the involved activities (mother and sons) is expressed by four methods conveying:

- the preconditions that must be satisfied by the mother activity to enable the change in status of some of the child activities and vice versa
- the post-conditions or effects of any change in status of a mother or child activity on the others.

Subsections 4.2–4.4 give specific properties defining non-primitive activities *before, iterate,* and *optional*. They also give an informal account of the content of the procedural slots.

### *4.2 Sequencing activities*

To specify that the activities A1, A2, …, An must be performed successively without any overlapping, one can use a *before* activity having A1, A2, …, An as child activities. The activity denoted by *before* (A, B) means that the activity B cannot have the status *open* before the status of A is *closed*. The time order of the sequence is expressed by the order of the list of constrained activities. Any *before* activity has two extra properties that allow, if necessary, specification of the delays between the opening of two consecutive activities, and between the closing of one activity and the opening of the next.

Among other preconditions, the first child must be allowed to change to *open* for the mother activity status to become *open*; similarly, for the last child to become *closed*, the mother must be allowed to change to *closed*.

The effect of a change in status of a *before* activity follows a set of rules, such as: “as soon as the mother changes to *open*, the first child changes to *open*” or “as soon as the last child activity changes to *closed*, the mother changes to *closed*”.

Another non-primitive activity 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.

### *4.3 Iteration*

An *iterate* activity, which has a single child (constrained) activity, specifies that the child activity should be repeated within the time during 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 replications, or any combination of the above possibilities. The child or descendant activities should not appear elsewhere in the plan. An *iterate* activity has two extra properties that allow specification, if necessary, of the delays between the opening of two consecutive iterations of the child, and between the closing of the child activity and the opening of its next iteration.

The only preconditions to a change in status of the child are that the mother be *waiting* or *open* for the child to change to *waiting*, and that the mother be *open* for the child to change to *open* or *closed*. As soon as the mother activity changes to *open* (resp. *closed*) the child changes to *waiting* (resp. *closed*) if possible.

As soon as the child changes to *closed*, it is set immediately to *waiting* unless the mother's closing conditions are satisfied at that time.

The iteration procedure, which is invoked each time the child activity changes to *closed* provided the mother is *open*, duplicates (instantiates) the child activity as needed in compliance with the constraints of delay between repetitions and limitations of the number of iterations, if provided. The child activity status is initialised to *waiting*.

### *4.4 Optional activity*

An *optional* activity has a single child activity and expresses that if this activity cannot be executed (i.e., if it is too late with respect to the opening interval or if the opening predicate cannot be satisfied), it is not a sufficient reason for the plan to be declared invalid. In other words, *optional*(A) means that the activity A should be executed 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* only if the child can change to *waiting*. Analogous preconditions apply when substituting *waiting* by *open* or by *closed* and by swapping 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 the mother activity cannot be executed, its status is forced to change to *closed*.

### *4.5 Updating the status of the activities in a plan*

The advance of time and the evolution of the production system (the biophysical system, in particular) may render true the opening and closing conditions of activities. The status of the activities is updated by a process responding to events scheduled to occur at any examination time specified by the manager (typically at discontinuity points induced by a new day or a new week) or on termination of an operation. The updating process invites the manager to run his own so-called Update method that essentially checks that the opening or closing conditions can be satisfied and that the constraints linking this activity to others would be satisfied if the change proceeded. When applied to the plan, this method causes a recursive examination of all the activities that are *waiting* or *open*. Any activity whose change in status is validated is updated, and the change is propagated immediately to the connected activities. See Martin-Clouaire and Rellier (2005, 2006), for a more formal presentation of this Update algorithm.

Normally, the status updating process is repeatedly invoked until the plan is closed. In some cases, the plan cannot be closed. Such a failure is detected when an activity that is not optional can no longer be opened or when it cannot be closed without violating restrictions induced by other activities (e.g., a *meet* activity in which the second child cannot be *open* although the first should be *closed*).

The Update procedure applies to an argument activity, which is initially the plan itself, each time the process is triggered. It uses opening and closing predicates attached to this argument activity. These predicates, which are specific to each non-primitive activity as shown in Sections 4.2–4.4, return true if it is legal to open or close the activity. They check whether preconditions of a change in status are satisfied or not. In case change is validated, Update calls a procedure that actually changes the status and propagates the effect to the connected activities as far as needed according to the activity-dependent rules.

## 5 Resources and allocation

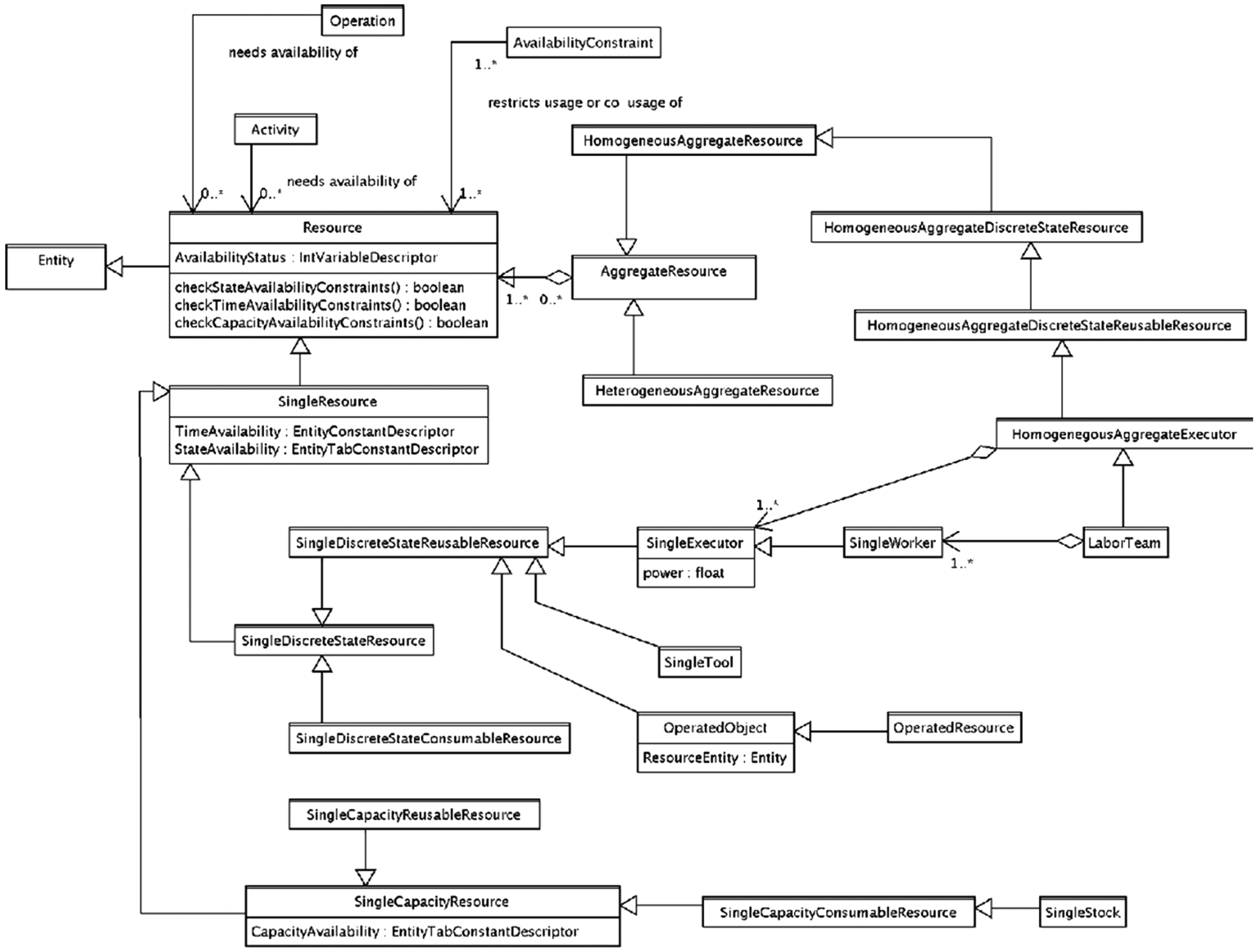
### *5.1 An ontology of resources and usage constraints*

Farm management is the process by which resources and situations are handled over time by the manager of the farm system in an attempt to achieve his or her goals. It is therefore essential to include the concept of resources in the ontology. Basically, a resource is an entity that supports or enables the execution of activities. Typically, the activity executors, the machinery involved and the various inputs (seeds, fertiliser, water, fuel) are resources. Resources are generally in finite supply and have significant influence on when and how activities may be executed. The availability of a resource is restricted by availability constraints that specify the conditions allowing their use or consumption. The constraints are temporal constraints (time windows of availability), capacity-related constraints (the amount available) or state-related constraints. Any resource is possibly constrained with respect to the maximum number of operations supported simultaneously and the maximum number of resources of other types that can be used simultaneously.

There are many types of resources that must be dealt with Smith and Becker (1997). A resource can be either consumable (usable only once) or reusable after it has been released. It can be a discrete-state resource (whose availability is expressed by a qualitative state such as *ready* or *not ready*) or a capacity resource (whose availability is characterised by a vector of numerical values expressing a multi-dimensional capacity). We distinguish between single resources and aggregate resources, which are collections of resources. See Figure 4 for a class diagram description of the part of the ontology dealing with resources.

In a primitive activity, the role of resource is played by the operated object, the operation resources and the executor. An operated object is a discrete-state resource that is a part of the biophysical system (an entity or a set of entities of the biophysical system). It is characterised by its ability to be transformed by several operations simultaneously. It may allow several resources to be simultaneously involved in transformations, and several executors to carry out certain transformations simultaneously.

**Figure 4** UML class diagram of the notion of resources




An operation resource is either a discrete-state resource (e.g., tools) or a capacity resource (e.g., diesel fuel). It is characterised by its ability to be used simultaneously for several objects acted upon in the biophysical system, to be involved simultaneously in several operations, and to be used simultaneously by several executors.

An executor is a discrete-state resource characterised by its (his) ability to work simultaneously on several objects in the biophysical system, to be involved simultaneously in several operations, to cope with several operation resources used simultaneously in the operations it (he) is engaged in. Another feature of an executor is its (his) work power that has an effect on the speed of the operation and on the requirement of operation resources if the latter are declared proportional to power. An executor is either an individual resource (e.g., a worker) or a labour team (a set of individual workers whose work power is by default the sum of the powers of the individual workers it comprises).

As an illustration, consider a cutting activity having the resource specifications shown in Table 1.

**Table 1** Resource requirements in a cutting activity

| *What is specified:* | *Specification:* | *Instances of entities or resources* (*): |
|---|---|---|
| Operated objects | "non-grazing fields greater than 0.5ha" | FIELD: {f1, f2, f3, …} |
| Operation resources | "one mower and one tractor" | MOWER: {m1, m2} TRACTOR: {t2} |
| Executors | "one person from farmer's sons or his employees" | SON: {s1, s2, s3} EMPLOYEE: {e} |

(*): small capitals refer to classes, normal characters refer to existing instances of the class.

The operated object specification refers to a set of spatial entities that are dynamically generated by expanding the entity set specification defining this set. Considering it as a resource is useful in case it is decided to disallow two simultaneous operations on any of these entities. The specification of resources coming with the operation component states that two machines are required: a mower and a tractor. The executor is a person to be selected either from the farmer's sons or his employees. If we have instances available in each of these classes, we have to consider two alternative allocations. In this example, at the time of allocation, the allocation procedure would return two alternatives {(f1, m1, t2, s2), (f1, m1, t2, e)} if f1 is the only field satisfying the request, m1 and t2 are the mower and tractor that are available, and s2 and e are, respectively, the second son and the employee who have no duty at that time. It might return a set of only one collection of assignments if no son or employee is available. It might of course return no solution at all, meaning that it is impossible to execute the activity immediately.

The use of resources is restricted by various constraints that make resource allocation a tricky combinatorial task. In addition to availability constraints, the ontology makes it possible to specify co-usage restrictions that concern the simultaneous use of a resource in different operations and combined with other resources. These co-usage restrictions are defined as specific entities having a slot whose value is a set (conjunction) of inconsistency conditions defined as cardinality limitations. The restriction called *activity-inconsistency-conditions* applies to an activity whereas the one called *resource-sharing-violation-conditions* applies to a resource. Finally, a third type of usage restriction called *activities-resources-inconsistent-commitments* is available in the ontology. It has two slots whose values are a set of *activity-inconsistency-conditions* and a set of *resource-sharing-violation-conditions*.

Table 2 shows an example of each type of usage restriction entity. The *activity-inconsistency-conditions* entity specifies that it is forbidden for any of the farmer's sons to use a tractor to cut a field. Checking this constraint amounts to making cardinality verifications relative to the number of sons and the number of tractors involved in the activity.

**Table 2** Example of restrictions on resource usage

| *Kind of constraint entity that is specified:* | *Specification:* |
|---|---|
| activity-inconsistency-conditions | {CUTTING; ((SON > 0)(TRACTOR > 0))} |
| resource-sharing-violation-conditions | {LOCATION; ((EXECUTOR > 1))} |
| activities-resources-inconsistent-commitments | { {GRAZING; ((DAIRY-HERD > 0)) } { LOCATION ; ((PESTICIDE > 0)) }} |

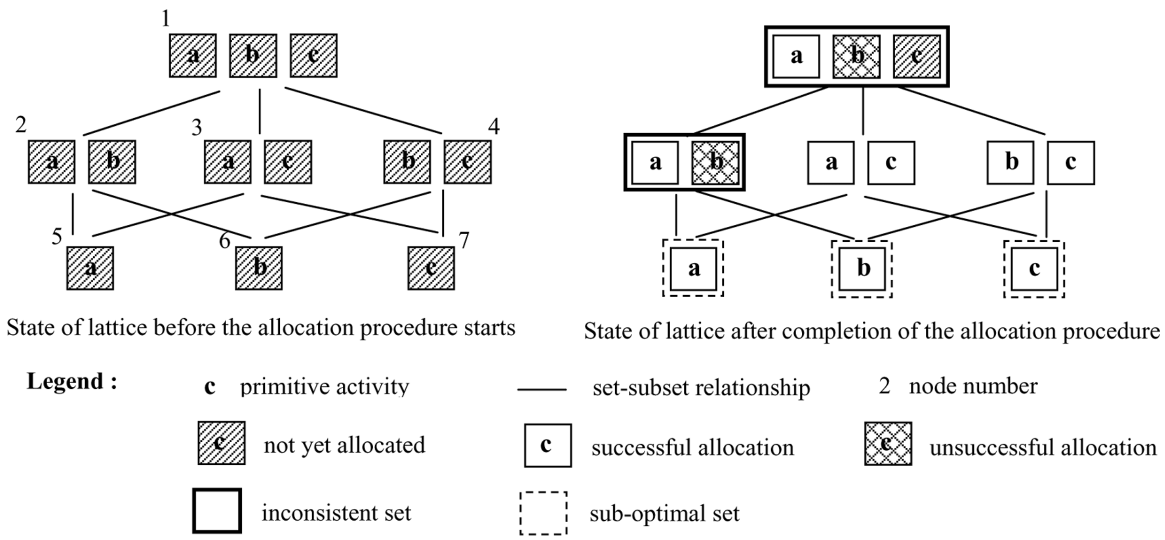
The *resource-sharing-violation-conditions* restriction states that there cannot be more than one executor at any location. Again, the checking of the constraint is a matter of cardinality verification, but this time, all allocations made so far for the current activity list are considered.

The *activities-resources-inconsistent-commitments* constraint states an incompatibility between a grazing activity by any dairy herd and the concomitant use of any pesticide at any location.

## 5.2 Resource allocation

This subsection outlines the main ideas implemented in the process that carries out the allocation of resources. Assume that at the current time we have to allocate a set of three primitive activities {*a*, *b*, *c*} that are eligible for execution according to the plan. The algorithm operates on a lattice of subsets of activities as shown in Figure 5. The nodes of the lattices are gone through in breadth-first order, starting from the top.

**Figure 5** Lattice of activities




In a given node, the algorithm tries to allocate each primitive activity in turn and propagates information incrementally in the lattice so that the search space can be significantly reduced. More specifically, if an activity is successfully allocated, the algorithm propagates this allocation to descending nodes that have the same starting activities up to the current one (e.g., node 1 and node 2 have the same starting activities up to *b*). In the example of Figure 5, the resources allocated to activity *a* in node 1 can be propagated to the same activity in nodes 2, 5 and 3. If a failure is encountered in an attempt to allocate an activity in a node, the node is declared inconsistent and all descending nodes having the same starting activities (up to the current one) are also declared inconsistent. For instance, the activity *b* in node 1 cannot be allocated once the activity *a* has been allocated; node 1 is declared inconsistent (no need to visit *c*), which cause node 2 to be declared inconsistent too. If a node becomes fully allocated, the descending allocated nodes are marked 'suboptimal' in the sense that they are included in larger set of executable activities. Once node 3 is fully allocated in Figure 5, the node 5 and 7 are marked suboptimal. Note that, due to the co-usage restrictions, the set of resources allocated to an activity (e.g., *c*) in a node (e.g., node 3) at a given level may be different from the set of resources allocated to the same activity in another node (e.g., node 4) at the same level.

At the end, the algorithm returns all the nodes that are fully allocated and neither suboptimal nor inconsistent, that is, nodes 3 and 4 in the example. The algorithm is complete in the sense that all solutions are produced. The choice of the one to be executed results from the computation of a scoring method that combines various preferences.

## 6 Example

The concepts and procedures defined in the above sections have been used to describe a grassland-based livestock production system (Martin et al., 2008). A challenge for such systems is the efficient and sustainable use of the perennial species-rich grasslands of the farmland to reduce herbage loss and fulfil the livestock feeding requirements. Different grassland types have different uses including the kind of grazing animals they are suitable for and the number of hay-making harvests they can sustain. They also have different growth patterns. The management problem is quite complex and is addressed in different ways by farmers, especially in their handling of climatic uncertainty, a recurrent difficulty for which they seek support and advice. Currently, simulation models of different instances of such systems are developed to enable researchers to analyse existing management practices and to design new ones. The simulation results are the basis of discussions between scientists and extension services.

The ontology has been extended (particularised) to entities such has fields or herds, activities such as grazing, feeding livestock with hay or concentrates, and cutting fields. Making a model of an executable plan is a matter of instantiating the abstract primitive activities provided as a library and articulating them using non-primitive activities. Figure 6 illustrates the kind of plans considered in this application.

**Figure 6** A plan of a grassland-based livestock system (see online version or colours)

```
1  and(
2      meet(  include(   meet(  grazing({F10}, Herd70),
3                                 optional(   grazing ({F11}, Herd70))),
4                          iterate   (hay-feeding(Herd70))),
5             and(    iterate( grazing ({F19A, F19B}, Herd70)),
6                     cutting({F10}, Farmer),
7                     cutting({F11}, Farmer))),
8      meet(  include(   grazing ({F3}, Herd24),
9                          iterate (hay-feeding(Herd24))),
10            and(    iterate( before( grazing({F5, F7}, Herd24),
11                                      or( grazing({F13}, Herd24),
12                                          cutting({F13}, Farmer)))),
13                    cutting({F3}, Farmer))),
14     meet(  include(meet(  grazing({F1, F2}, Herd21),
15                              optional(grazing ({F4}, Herd21))),
16                    iterate(hay-feeding(Herd21))),
17            and(    equal(  iterate( grazing({F11A, F11B, F11C }, Herd21)),
18                              iterate (concentrate-feeding(Herd21))),
19                    cutting({F4}, Farmer),
20                    optional( cutting({F1, F2}, Farmer)))),
21     and(   cutting({F6}, Farmer),      cutting({F8}, Farmer),
22            cutting({F9}, Farmer),      cutting({F12}, Farmer),     cutting({F14}, Farmer))
23 )
```

The opening and closing conditions of the activities have been omitted for the sake of brevity. Typically they refer to herbage availability on the fields or the physiological stage of grass. The *and* activity in the first line makes it possible to wrap four activities. The first three ones are *meet* activities and the last one is another *and* activity. Only the first one (lines 2 to 7) is commented on here. The primitive activities are written in grey characters like the following: operation (operated-object, executor) or operation (operated-object) when there is no need to specify an executor component. The entity names starting with F (resp. H) denote fields (resp. herds). Note that the operated object component may be a collection of entities as in the grazing activity in line 5 where two fields (F19A and F19B) are specified.

The *meet* activity in line 2 constrains two composed activities, the first one (lines 2 to 4) being an *include* activity, and the second one (lines 5 to 7) an *and* actvity. Semantically, *meet* specifies that its two constrained activities are contiguous: as soon as the first one ends, the second one is activated. Similarly, the *include* activity constrains two activities. Semantically, it forces the period during which the second one has *open* status to be included in the period during which the first one has the same status. The two activities constrained by *include* are two composed activities, the first being a 'meeting' sequence of two grazing activities (one required and one optional) and the second being an iterative hay-feeding activity. The *and* activity in lines 5 to 7 wraps three activities: a grazing activity to be iterated and two cutting activities.

## 7 Related works

Several agent behaviour 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 developed primarily to support formal reasoning about current and potential agent activities to ensure that certain properties are complied with. ConGolog allows specification of complex plans that are kinds of control procedures. The main difference with our approach is that our interpreter can only determine repeatedly the actions that are eligible for execution; non-executability is a property that is eventually revealed when a dead end is met. Actually, for the target applications, we are more interested in a probabilistic assessment of the non-executability of a plan; 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 revealed by simulation calls for modification of the plan or of the conditional adjustments that should be included in the management strategy for providing plan adaptation capabilities. In addition, we address management problems that involve rich temporal and procedural constraints on and between activities. We have paid special attention to making the plan intelligible through the language. The actions have complex and highly uncertain consequences that are difficult to incorporate in an action theory intended to allow reasoning about their anticipated effects.

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 organisation of actions. They have an execution procedure capable of implementing open-ended responsive decision-making behaviour based on high-level control constructs. However, these languages do not offer rich ready-to-use primitives to express temporal constraints on the activities. Consequently, it is hard to reproduce the ability to maintain a sense of continuity in the application of a nominal plan. Neither the PRS nor the ConGolog types of model have primitives dedicated to the management of resources.

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

Finally, other languages have been developed to model and simulate work processes. Among them is the multiagent environment Brahms (Sierhuis et al., 2007), developed by NASA and geared towards modelling people's activity behaviour in space missions.

## 8 Concluding remarks

We have presented a special-purpose work organisation language developed for modelling agricultural production tasks that are highly dependent on uncontrollable exogenous factors and that involve activities constrained by rich temporal properties and resource requirements. As pointed out in the previous section, the problem of developing purposive programmable action behaviours in open environments is also addressed by the planning/scheduling and autonomous agent communities in AI. In these approaches, the emphasis is more on the automatic construction of plans and formal verification of plan properties or on execution performance. Because we only aim at simulating decision behaviour, we give greater importance to the development of a rich representation language that can incorporate the kind of knowledge used by production managers in practice. The language must allow sufficient flexibility, so that premature decisional commitment can be avoided, and plan-based reasoning and resource allocation can be interleaved at the time of execution.

The framework is quite generic. It might be applicable in other domains than agriculture but we did not attempt to do so. Moreover, it is likely that other domains such as manufacturing may not need the same kind of features than those introduced to deal with the uncertainty around driving factors such as weather. In manufacturing, uncertainty affects what needs to be produced (the demand) rather than the production process itself (Martin-Clouaire and Rellier, 2006).

Our ontology, together with the simulation environment that implements it, provides assistance in the development of production system simulation models by guiding the knowledge elicitation process and by minimising the amount of code to be written. In developing a farm production system model, the ontology acts as a metamodel; implementing a model amounts to particularising the ontology concepts as required by the domain and then instantiating the corresponding classes to capture the specific aspects of the system to be simulated.

To be used and shared, ontology has to be consensual, concise, precise and encompassing. To tend towards these properties in the work practice ontology, we have interacted with farming system experts and imported notions long used in the workflow and in production management communities (WFMC, 1996). However, in this paper, we do not claim to have developed the ultimate ontology capable of capturing the whole essence of work organisation knowledge. Actually, a strong point of the ontology is its extensibility, thanks to the generic nature of the underlying framework. An extension currently under development concerns terms such as goal, preference and inferential mechanisms required, for instance, to model anticipation in decision processes. A combination with the Belief-Desire-Intention (BDI) type of decision-making architecture (Rao and Georgeff, 1995) is being considered. Beliefs express the manager's current state of knowledge about the production system; intentions are the activities structured in a plan; desires are specifications about dated target states of the production system. Another extension addresses the modelling of spatial features and their dynamics.

The ontology and DIESE, its associated simulation framework, are currently used in two large projects: the one briefly considered in the example in Section 6, and MELODIE (Chardon et al., 2007), which involves farm models integrating crop, dairy and pig production systems. The development of the ontology was largely inspired by the analysis made in a modelling project on greenhouse tomato production systems (Jeannequin et al., 2003).

## References


Booch, G., Rumbaugh, J. and Jacobson, I. (2005) *The Unified Modeling Language User Guide*, 2nd ed., Addison-Wesley Professional, Boston.

Carberry, P.S., Hochman, Z., McCown, R, Dalgliesh, N., Foale, M., Poulton, P., Hargreaves, J., Hargreaves, D., Cawthray, S., Hillcoat, N. and Robertson, M. (2002) 'The FARMSCAPE approach to decision support', *Agricultural Systems*, Vol. 74, No. 1, pp.141–177.

Chandrasekaran, B., Josephson, J. and Benjamins, V. (1999) 'What are ontologies and why do we need them', *IEEE Intelligent Systems*, Vol. 14, No. 1, pp.20–26

Chardon, X., Rigolot, C., Baratte, C., Le Gall, A., Espagnol, S., Martin-Clouaire, R., Rellier, J-P., Raison, C., Poupa, J-C. and Faverdin, P. (2007) 'MELODIE: a whole-farm model to study the dynamics of nutrients in integrated dairy and pig farms', in Oxley, L. and Kulasiri, D. (Eds.): *MODSIM 2007 Int. Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand*, December, pp.1638–1645, ISBN: 978-0-9758400-4-7, http://www.mssanz.org.au/MODSIM07/papers/25_s25/MELODIE_s25_Chardon_.pdf

Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P. (1998) *Open Knowledge Base Connectivity 2.0*, Report of Knowledge Systems Laboratory, Stanford, CA.

De Giacomo, G., Lespérance, Y. and Levesque, H. (2000) 'Congolog, a concurrent programming language based on the situation calculus', *Artificial Intelligence*, Vol. 121, pp.109–169.

Ingrand, F., Georgeff, M. and Rao, A. (1992) 'An architecture for real-time reasoning and system control', *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, Vol. 7, No. 6, pp.34–44.

Jeannequin, B., Martin-Clouaire, R., Navarrete, M. and Rellier, J-P. (2003) 'Modelling management strategies for greenhouse tomato production', *Proc. CIOSTA-CIGRV Congress*, Turin, pp.506–513.

Martin, G., Duru, M., Martin-Clouaire, R., Rellier J-P., Theau, J-P., Thérond, O. and Hossard, L. (2008) 'Towards a simulation-based study of grassland and animal diversity management in livestock farming systems', *Proc. iEMSs2008*, Barcelona, Spain, Vol. 2, pp.783–791, http://www.iemss.org/iemss2008/uploads/Main/Vol2-iEMSs2008-Proceedings.pdf

Martin-Clouaire, R. and Rellier, J-P. (2005) 'Representing and interpreting flexible production management plans', *Proc. Conceptual Modelling and Simulation Conf. (CMS 2005)*, Marseille, pp.69–76, F. http://carlit.toulouse.inra.fr/diese/docs/CMS05.pdf

Martin-Clouaire, R. and Rellier, J-P. (2006) *Fondements ontologiques des systèmes pilotés'*, Internal report UBIA-INRA, Toulouse-Auzeville, F. http://carlit.toulouse.inra.fr/diese/docs/ri_ontologie.pdf

Morley, D. and Myers, K. (2004) 'The SPARK agent framework', *Proc. AAMAS-04*, New York, pp.712–719.

Rao, A. and Georgeff. M. (1995) 'BDI agent: from theory to practice', *Proc. Int. Conf. on Multiagent Systems*, San Francisco, pp.312–319.

Rellier, J-P. (2005) *DIESE: un outil de modélisation et de simulation de systèmes d'intérêt agronomique*, Internal report UBIA-INRA, Toulouse-Auzeville, http://carlit.toulouse.inra.fr/diese/docs/ri_diese.pdf

Sierhuis, M., Clancey, W.J. and van Hoof, R.J. (2007) 'Brahms: a multi-agent modelling environment for simulating work processes and practices', *Int. J. Simulation and Process Modelling*, Vol. 3, No. 3, pp.134–152.

Smith, M.K., Welty, C. and McGuinness, D. (Eds.) (2004) *OWL Web Ontology Language Guide*, http://www.w3.org/TR/owl-guide

Smith, S.F. and Becker, M.A. (1997) 'An ontology for constructing scheduling systems', *Proceedings of the AAAI Spring Symposium on Ontological Engineering*, April, Palo Alto, CA, pp.120–129.

Smith, S.F., Hildum, D.W. and Crimm, D.R. (2005) 'COMIREM: an intelligent form for resource management', *IEEE Intelligent Systems*, Vol. 20, No. 2, pp.16–24.

WFMC. (1996) *Workflow Management Coalition Terminology and Glossary, (WFMC-TC-1011)*, Technical Report, Workflow Management Coalition, Brussels.