

A methodology for building agent-based simulations of common-pool resources management: from a conceptual model designed with UML to its implementation in CORMAS

C. Le Page and P. Bommel

Since 1995, our team has been developing a simulation platform called CORMAS (common-pool resources and multi-agent systems). It provides facilities to build and analyze agent-based models (ABMs) that represent ecosystems where various human activities compete for the use of natural resources. Few agent-based simulations can be mathematically proven, but they can be analyzed inductively. It is therefore important that simulations be replicated before they are accepted as correct. To tackle this thorny issue of ABM replication, we believe that, during the design process, a careful representation of the conceptual model is paramount. In this paper, we advocate using UML (unified modeling language), which is a formal language to describe systems using the object-oriented paradigm. An archetypical agroforestry system is presented here, and serves as an example to design a very simple model dealing with common-pool resources management. Different types of UML diagrams are also introduced to describe the static structure of the model, as well as that of the dynamic processes. Adaptation of these diagrams for implementation using the CORMAS platform is detailed. Then, a simple simulation scenario is presented to illustrate how it is done in CORMAS, and a sensitivity analysis on one parameter of the model is conducted.

Common-pool resources (CPR) management involves interactions among stakeholders and groups of stakeholders in using the resources. It is about decision making in space. In the process of decision making, each individual stakeholder tries to achieve a personal goal, and at the same time may also be constrained by some regulations or rules established at a collective level (group or institution). As natural resources are most often heterogeneously distributed, this will influence the actions of several stakeholders on these resources. For instance, each user will determine strategic places according to some specific criteria. But access rules, a key issue in CPR, may prevent the achievement of some activities. Access rules are often related to the spatial characteristics of the environment, such as slope, distance, elevation, adjacency, connectivity, etc. Hence, it is crucial to take into account the spatial aspects of CPR management.

Agent-based models (ABMs) are particularly well suited to represent ecosystems where contrasting human activities compete for the use of natural resources in space. ABMs are based on the principles of multi-agent systems (MAS), a research

field in computer science focusing on distributed artificial intelligence. An agent is a virtual entity, a computer component, such as a software (program) or a hardware (robot), that is driven by individual objectives, capable of perceiving its surrounding environment and capable of acting on its environment, and that can also communicate directly with other agents (Ferber 1999).

Using MAS to investigate how CPR can be managed is fast becoming a research field. From a theoretical point of view, ABMs of individual decision making have been studied (e.g., Jager et al 2000) and, recently, Deadman and Schlager (2002) have reviewed their use in the specific context of common-pool-resource management institutions. Since 1995, the Green research unit from the French Agricultural Research Center for International Development (CIRAD) has been developing a simulation platform, CORMAS (**common-pool resources and multi-agent systems**), which views CPR from a more generic and practical perspective (Bousquet et al 1998). Our seminal objective was to be able to design models more easily, rapidly, and efficiently based on interactions between natural and social dynamics in the context of CPR management. Today, among the existing agent-oriented simulation platforms (Gilbert and Banks 2003), CORMAS remains very open by not imposing any predefined individual decision-making process on an agent, or coordination protocols between agents. This flexibility also leaves the responsibility to describe all the details of the model to the model designer. Moreover, according to the scientific reproducibility principle, it should be possible for anybody with basic skills in modeling to build the model again, to reimplement it by using any appropriate simulation toolkit (not necessarily the one used originally), and to verify that the results obtained are the same as the ones originally published. This is very challenging as the model becomes complex. Today, it is one of the biggest concerns of scientists from the fields of social science, economics, and ecology in using ABMs to simulate artificial societies or ecosystems (Hales et al 2003).

To ensure a rigorous description of a particular ABM, providing the source code appears to be a necessity, but it is definitively not sufficient. In between the literal description of a model and its implementation in a computer using a specific programming language, a formal representation of the conceptual model is vital. Recently, a standard methodology, UML (unified modeling language), has emerged (Bergenti and Poggi 2002). Recently the Foundation for Intelligent Physical Agents (FIPA) has even proposed a specific extension of UML toward multi-agent systems.¹

Our objective is to present a methodology for designing an ABM with CORMAS through a formal UML representation of the corresponding conceptual model. A simple but complete model will help to illustrate what the UML represents and how to run models with CORMAS. The scope of this paper is more about how to design an ABM with CORMAS, rather than about the substance of the model. Hence, before describing the toy-model, we will first introduce the formal concepts used in the set of basic UML diagrams, as well as the related conventional notations. Second, we present an archetypical model of CPR management, the slash-and-burn toy-model. A literal description of the model is proposed, followed by the conceptualization of

¹www.auml.org.

the model using UML. Third, we present the implementation of the conceptual model with CORMAS and propose a set of simulations.

UML overview

The “unified modeling language” (UML) is a description language, specifically a graphic-based representation language of models. It is an open tool designed to be independent of particular programming languages (such as Java or Smalltalk). UML is a formal and normalized language and was accepted by the OMG (Object Management Group) in 1997 (OMG 2003a,b). From then on, UML is the reference in terms of object modeling: a universal language for object-oriented languages. The specifications of the most recent official version (1.5) are available from the OMG Web site.²

This paper is dedicated to modelers and scientists willing to build ABMs on a framework such as CORMAS,³ Swarm,⁴ RePast,⁵ etc. Whatever the targeted platform, the UML diagrams are used to explain a model and they have to be independent from the platform and the computer language. Indeed, an ABM described with UML is an abstract representation that gives a simplified picture of the real world. Because UML is based on simple graphic notations, with UML diagrams, an ABM should be understandable even by noncomputer scientists. UML can be seen as a dialogue tool that should facilitate communication among scientists, modelers, and stakeholders. Our goal here is not to review all the formal aspects of UML, but at least to give useful insights for nonspecialists who may be interested in using UML to specify ABMs.

Formalizing the structure of a model using the UML class diagram

The UML class diagram is the basic building block for conceptual modeling. It shows all the classes (or a part) and their relationships that are relevant for the purposes of the phenomenon to be modeled. Drawing the class diagram is the first and the main stage of the modeling process. This stage is particularly fruitful when it takes place during a collaborative working session.

Creating a simple and understandable class diagram can be a long and difficult process. In practice, the first step consists of identifying the relevant real-world types of entities and then mapping out each of them using the concept of *class*. A class can be considered as a description of objects having a similar structure and similar behavior and sharing a common semantic. Practically, a class is defined by a list of characteristics (called “attributes”) and a list of behaviors (called “operations”). Attributes represent the static part while operations represent the dynamic part. A class can also be viewed as the “generator” of the objects (called “instances” of the class). In other words, a class describes a structural model for a set of similar *objects*, called instances of this class (see Fig. 1).

²Pending issues for UML specification are available from the OMG official Web page: www.omg.org/technology/documents/formal/uml.htm.

³<http://cormas.cirad.fr>.

⁴<http://wiki.swarm.org>.

⁵<http://repast.sourceforge.net/>.

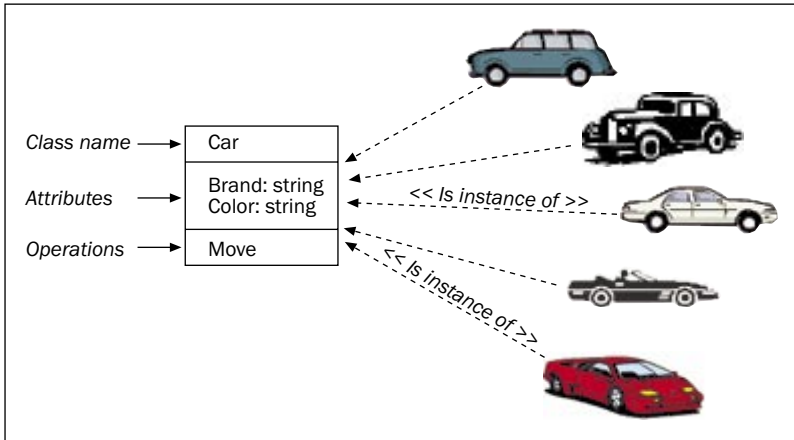


Fig. 1. Class and instances. The five cars on the right are the objects. Even if they are different, they belong to the same concept, the Car class, which has two attributes (brand and color) and one operation (move).

An object encapsulates its data in order to better control any modifications. Indeed, as seen from the outside (i.e., by the other objects), an object shows only its external interface, which is a set of *public* operations. UML specifies visibility of operations and attributes with markers before the names: “+” stands for public and “_” for private. Mostly, an attribute is private: nobody except the object itself can directly access its value and change it. Conversely, a public attribute can be accessed and modified by other objects. If necessary, two corresponding accessing operations are defined. The “reader”-accessing operation returns the value of the attribute and the “writer”-accessing operation allows one to change the value.

UML promotes a development process that is iterative and incremental. According to a standard software life cycle,⁶ UML proposes different types of class diagrams. In a class diagram at the “analysis” stage, many details are omitted, such as visibility, types of attributes and types of values returned by operations (if any), and parameters (arguments) of operations (if any). The same diagram at the “design” stage introduces all these details. Conventionally, the types are indicated after a colon, and the potential arguments of operations are indicated between parentheses (see Fig. 2).

Relationships between classes are called *associations*. Associations are drawn as straight lines between the two rectangular boxes representing the classes. Usually, an association is denoted by a verb describing its semantic. The extremities of an association should indicate its *multiplicity* (an integer value or a range of integer values) and its role (a string label) played by the related class in the context of the association. Additional *comments* are shown as text strings (not enclosed in parentheses) within a note icon directly linked to the related element to be commented.

To make such abstract notions clearer, let us formalize with UML a pattern commonly used in the field of renewable resources management. Imagine a portion

⁶The main stages of a software life cycle are (1) “analysis,” (2) “design,” (3) “implementation,” and (4) “tests and maintenance.” Mostly, an analysis diagram is sufficient to describe the structure of an ABM.

of land covered by a land cover with a biomass that grows up according to the standard logistic equation. Figure 2 shows the corresponding class diagram at the design stage.

Two classes are defined, LandUnit and LandCover. These two classes are connected through an association semantically understandable as “a land unit is covered by a land cover.” From a land-unit perspective, the associated land cover may be simply called *cover*. This is a role played by a land cover through the eyes of a land unit. Symmetrically, from a land-cover perspective, a land unit may be seen as a *place*. By drawing the number 1 at both extremities of this association, we state that a given land unit is covered by exactly one land cover (i.e., a land unit without land cover makes no sense in this context), and reciprocally a given instance of land cover is located in exactly one land unit. Figure 2 contains another example of association, which is a bit particular as it is reflexive. Associations in UML express interactions between agents in multi-agent systems. Reflexive associations express interactions between similar entities. Here, the connection between any particular land unit and its four neighbors depicts the structure of a standard “von Neumann” cellular automata network. ABMs dealing with renewable resources management are frequently using such a structure to represent the environment.

In UML, “underlining attributes” means to give them a special status. An underlined attribute corresponds to a “class variable,” whose value is specific to the class itself and therefore will be the same for all the instances. Returning to Figure 2, we can interpret the diagram for the LandCover class. Every instance of land cover has a biomass, but two different instances may have two different values of biomass. The same reasoning could not be applied for the intrinsic growth rate r and the carrying capacity K (the two parameters of the logistic equation). Two different instances of the same kind of land cover should share the same values for r and K , as if they belonged to the LandCover “species”. Then, the *growth* operation will be a matter of updating the value of the *biomass* instance variable, by referring to the previous value of the *biomass* attribute and to the two class variables r and K . Moreover, in UML it is possible to indicate values. In Figure 2, we can see that r is a float equal to 0.4 and K is an integer equal to 1.

Associations starting with a lozenge are simple associations with the special semantic “is made of” (“is aggregated from”). The multiplicity is represented by the symbol “1..*”, which means that a woodlot can be composed of at least one land-unit

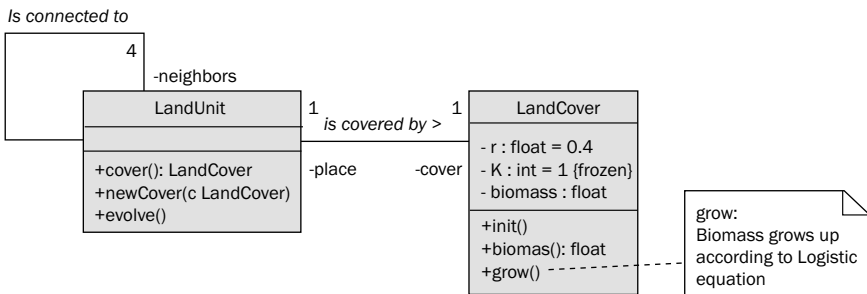


Fig. 2. UML land-use pattern (at design stage).

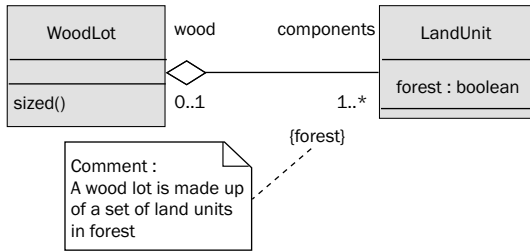


Fig. 3. Example of UML aggregation. A WoodLot is made up of forested land units. A forested land unit belongs to a WoodLot and plays the role of a component of this WoodLot.

instance up to any number of instances. For instance, in Figure 3, a woodlot is defined as an aggregate of (at least one) land unit respecting a constraint: being forested. In UML, constraints express conditions or limitations. They have to be written between curly brackets.

Now, let us present a quite different concept from association: *generalization* is an intellectual mechanism for either refining a concept (specialization) or abstracting a concept (generalization). This mechanism is a *second abstraction* level (after the notion of class regrouping similar objects). Generalization means relating several classes that have some properties in common to a more general “super class.” Thus, a specific class is a specialization of a more general class. As a corollary to that, a subclass inherits the features of its super class (attributes, operations, associations, and constraints). However, a subclass may redefine a part of the description that it “inherits.” Figure 4 represents a hierarchy of specialization for the LandCover class.

To better understand inheritance principles, let’s detail the *Pasture* class as it appears in Figure 4. Because a pasture is a kind of land cover, it inherits one instance variable (age) and three class variables (implantation cost, upkeep cost, and suppression cost). The values for implantation and upkeep costs are redefined. Because a pasture is also a kind of changing cover, it is characterized by one additional instance variable (neglected duration) and two additional class variables (transition age, whose value is redefined, and natural succession, which is also redefined at the level of Crop as a class association from Crop to Fallow, meaning that the next stage of a pasture will be a new instance of fallow). Finally, because a pasture is also a kind of crop, it has two more additional class variables (price per Kg and production per Ha), whose values are redefined.

Depicting model dynamics

Dynamics diagrams are common mechanisms for describing system evolution over time. In UML, several types of dynamics diagrams allow us to describe the behaviors of the entities and their interactions. Each type provides a slightly different capability that makes it more appropriate for certain purposes. We promote three types of representations for specifying the dynamics aspects of an ABM: activity diagrams (intra- or interobject dynamics), state-transition diagrams (internal dynamics of an object), and sequence diagrams (dynamics among objects).

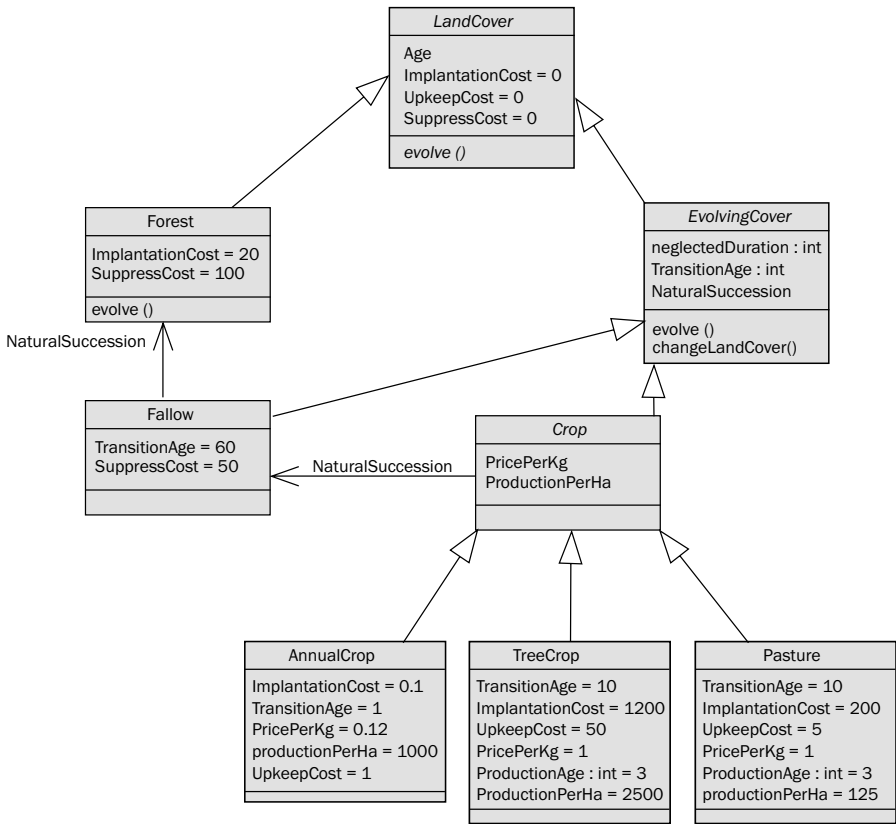


Fig. 4. Hierarchy of specializations for the LandCover class.

Sequence diagrams. The sequence diagram describes the sequence of messages that are exchanged among objects over time. These exchanges are shown along the objects' lifelines. An object's lifeline represents an instance of the class, that is, an individual participant in the interaction. Arrows between the lifelines denote communication between the instances. From top to bottom, the order of messages along a lifeline is significant, as it denotes the order in which these messages will occur. A message defines one specific kind of communication in an interaction. These communications are used to invoke an operation. In any parts of a UML sequence diagram, conditions (called "guards," which are enclosed by square brackets) can be used if necessary.

Discrete time-step schedulers (such as CORMAS) slice the time stream in homogeneous time-steps and activate the model objects sequentially. For example, a time-step duration can be equivalent to one year. Each year, the scheduler activates the model entities that perform their annual activities. To explain this regular sequence of activities, which can be interpreted as the dynamic part of a simulation scenario, a UML sequence diagram is suitable. Figure 5 shows a very simple sequence diagram for a model with nothing but an intrinsic dynamics of land-cover changes.

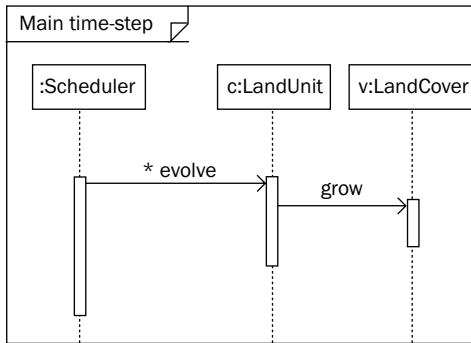


Fig. 5. A simple example of a UML sequence diagram.

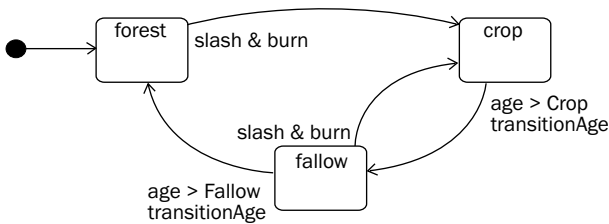


Fig. 6. A simple example of a UML state-transition diagram.

The scheduler sends the “evolve” message to a set of land units. The “*” character before the message name indicates that this message should be repeatedly sent to a set of instances. When a given instance of LandUnit receives this message, it is activated and in turn sends the “grow” message to its land cover.

State-transition diagrams. State-transition diagrams are used to describe the behavior of one object. They show the possible sequences of states through which an object instance can proceed during its lifetime as it reacts to events (for example, signals, operation invocations).

Figure 6 displays the three states that a LandUnit can have. A transition is crossed from one state to another when an event occurs. At this stage, the origin of this event is unknown. It may arise from internal activities (age higher than transition age) or from external actions (slash and burn).

The black dot represents a pseudo-initial state. It can be omitted; it just helps to fix the starting point to read the graph. The events are a kind of stimulus. They trigger the transition to the next state.

Activity diagrams. Activity diagrams are commonly called “control flow” and “object flow” models, and they can be seen as a revision of the standard flow-chart diagrams. The purpose of an activity diagram is to describe a set of activities by representing actions and their consequences. Actions can be described by natural language. A transition is a relationship between two activities indicating that an instance will enter the second activity and perform specific actions as soon as the previous activity has ended. When several kinds of instances are involved in the set of activities to

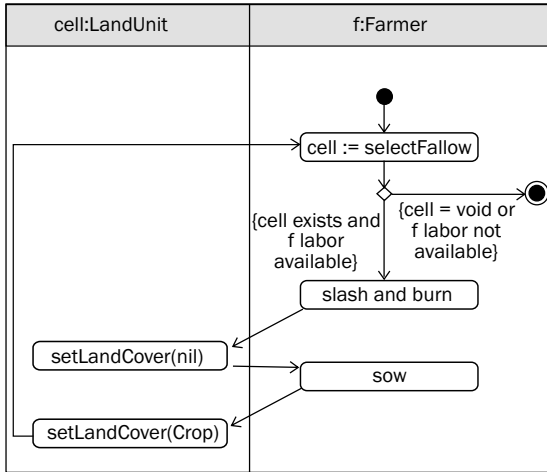


Fig. 7. A simple example of a UML activity diagram.

be described, “swim lanes” (one per instance) delimited by vertical solid lines are introduced. The relative ordering of the swim lanes has no semantic significance.

The activity diagram shown in Figure 7 represents a chain of activities between a land unit and a farmer.

At the first stage, the farmer selects a land unit covered by fallow. When this activity ends, a transition is fired to a “decision point” (a lozenge). According to the guard’s value, the main activity may finish or may enter into a loop (while the farmer’s manual labor is available and a land unit has been selected). This loop consists of several activities: the farmer slashes and burns the plots (we can suppose that this activity decreases the manual labor) and the cover of the land unit is removed and then the farmer sows and a new crop is implanted on the land unit. At the end of the loop, a new land unit is selected. The same activity of a farmer can be represented with another activity diagram (see following figure) that adds an “object flow” (Fig. 8). In parallel to the sequence of activities, an instance of LandUnit, called a “cell,” is shown through its different states.

A simplistic and archetypical model of CPR management: the “slash-and-burn” toy-model

To manage a common resource in a sustainable way, it is often asserted, referring to Hardin’s seminal paper about the “Tragedy of the Commons” (Hardin 1968), that some restrictions should be imposed by an authority on individual practices. The model presented here, called “slash and burn,” illustrates how the interrelated dynamics between individual and collective representations of a renewable resource may influence individuals reciprocally. It was inspired by a previous CORMAS model elaborated by a geographer, J.L. Bonnefoy (Bonnefoy et al 2000, 2001).

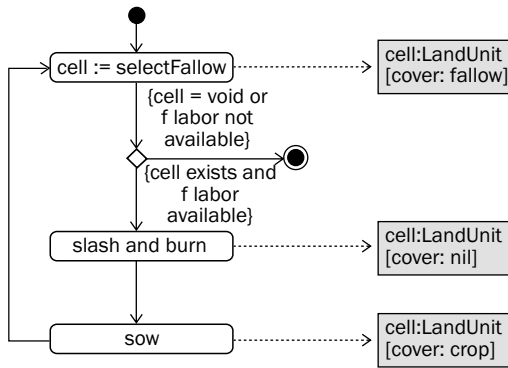


Fig. 8. A simple example of a UML activity diagram with object flow.

Literal description of the “slash-and-burn” model

A virtual forest landscape is set as a square lattice made up of 50 by 50 hexagonal LandUnits. Each LandUnit, which represents a homogeneous portion of space, may be covered by forest or not. For any LandUnit without forest, the chance of forest recovering is proportional to the number of neighboring LandUnits being covered by forest.

Some farmers are located on a given LandUnit. With each time-step, they move from the LandUnit where they are located to a neighboring one. With a limited perception range around their location, they can perceive the neighboring forest, if there is any. When they do not perceive any forest, they simply move randomly; otherwise, they decide whether or not they move toward a neighboring LandUnit covered by forest to “slash and burn” it.

A ForestDepartment is in charge of periodically organizing (not every time-step) a census of the forest resource by identifying, sizing, and marking patches of contiguous LandUnits being covered by forest (called WoodLots). Marks set on the WoodLots are about protection of the forest resource: if the size of a WoodLot is below the authorized minimum size, it will be marked “protected.” To determine the authorized minimum size, the ForestDepartment requests that all the farmers individually report each of their perceptions of the WoodLot’s mean size; the highest value among all the reported values is the authorized minimum size.

Individual farmers use a memory (with limited capacity to store) to remember the sizes of the WoodLots they have encountered; when their memory becomes full, they just forget about the less recent stored value. Two contrasting strategies of individual farmers (“conformist” and “nonconformist”) are illustrated using two factors affecting their decision-making process: the first factor is related to their reported value of the minimum size—for “conformist” farmers, this will be the arithmetic average of the recorded values; for “nonconformist” farmers, it will be the highest value from the recorded values. The second factor is related to the way a farmer decides whether he/she will slash and burn a perceived LandUnit covered by forest. A “conformist” will certainly respect the protection mark set by the ForestDepartment, whereas a “nonconformist” will refer to his/her personal computed average value—if the size

of the WoodLot is higher than this value, the nonconformist will decide to slash and burn the LandUnit even if it belongs to a WoodLot marked as protected.

Conceptualization of the “slash-and-burn” model using UML

Class diagram of the model at the analysis stage. Figure 9 represents a class diagram of the slash-and-burn model described earlier.

The left part of the class diagram describes the spatial aspect—the landscape is composed of two types of entities: the elementary level (LandUnit) and the aggregated level (WoodLot). The components of a WoodLot are instances of LandUnit that are connected and in a forest state (constraint). On the other hand, one LandUnit may belong to a WoodLot if its state is forest. The size of a WoodLot is the number of its components. A WoodLot can be declared protected or not.

When deforested, a LandUnit can recover its forest according to a probability (probaForestRecover). We assume here that this probability, equivalent to a recovery rate, is a constant value shared by any deforested LandUnit (it is then a class variable).

The landscape is made up of 2,500 LandUnits. A comment related to this multiplicity states that these elementary spatial entities are organized as a “50 × 50” square spatial grid.

A Farmer entity can be regarded as a composite entity. Indeed, farmers have their own inner state (perception range, etc.) and inner behavior (goSlashAndBurn or moveRandomly), but they also own a specific strategy that can change over time: they can be conformist or nonconformist. In this particular model, “Strategy” is an abstract class, meaning that its raison d’être is only to serve as a generalization of the two specific strategies. “Strategy” declares two abstract methods that are refined in both subclasses (“Conformist” and “NonConformist”). This object architecture is called *polymorphism*; it allows users to specify similar behaviors but can be carried out differently. This structure will be convenient for discriminating the two strategies. For instance, to state that, for the conformist, the reported value is the arithmetic average of the values stored in “cuttingMemory” (one of the attributes of the Farmer class, see Figure 9), and that, for a nonconformist, it is the highest value from the one stored in “cuttingMemory,” it is simply a matter of writing two different versions of the same method called “reportValue.”

Class diagram of the model at the design stage. Figure 10 represents the same slash-and-burn model, but in a detailed design stage. In this stage, more details are revealed, such as visibility, types of attributes, by-default values, and parameters of operations.

A lot of additional information is thus provided in Figure 10 compared with Figure 9. For instance, for attributes, it is indicated that the default value for the class variable probaForestRecover (LandUnit class) is set to 0.0025. For operations, if we look at the Farmer class, we can note from the “+” sign that “goSlashAndBurn” and “sendReport” are the two main behaviors available for outer use (public methods). The other operations are for private use: – perceive():LandUnit [*]) is a private method without argument that returns a set of LandUnits.

The sequence diagram. The sequence diagram shows the basic order of a series of operations in a simulation. The sequence diagram in Figure 11 shows how the

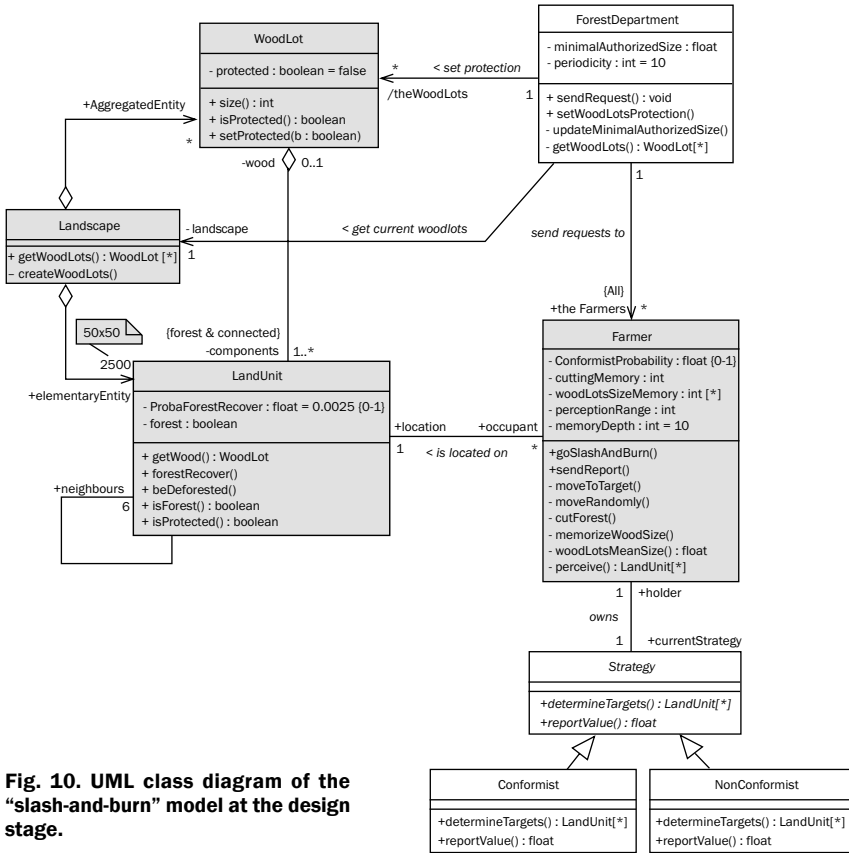


Fig. 10. UML class diagram of the “slash-and-burn” model at the design stage.

scheduler activates the entities of the slash-and-burn model. Each year (assuming that a time-step is equivalent to one year), the scheduler activates the LandUnits for forest recovering, then activates the farmers to perform their annual activities and the ForestDepartment to set the WoodLotsProtection.

In drawing the sequence diagram, a modeler has to take note of the risk in representing all the possible message exchanges over time; this can lead to an incomprehensible diagram that defeats the purpose of UML. A sequence diagram should be restricted to the main operations that are triggering the internal behaviors of each entity of a model, and therefore should avoid delving into any internal details of such or such operations. Rather than producing a single but complicated sequence diagram, a better solution would be to restrict it to its simple expression, as in Figure 11, and to associate it with other sequence diagrams (Fig. 12) or activity diagrams (see Fig. 14).

A specific internal periodicity of activities exists for the ForestDepartment. On the left of the activity lifeline of the ForestDepartment, a guard condition (between square brackets) depicts this specific periodicity.

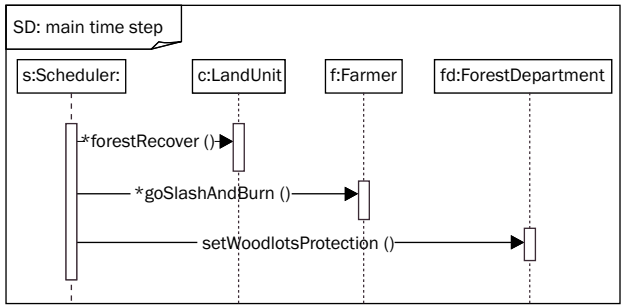


Fig. 11. UML sequence diagram of the main step of the scenario defined in the “slash-and-burn” model.

The WoodLots can be considered as a reification of a point of view. They are the minimum spatial unit in the eyes of the ForestDepartment. Unlike the LandUnits, which are created during model initialization, some WoodLots can also be created during the run time. To have a reference on them, the ForestDepartment asks the Landscape to identify them, which means creating new WoodLots from the LandUnits according to the constraint { forest & connected }. “getWoodLots()” is the only public operation of the Landscape class (see Fig. 9).

State-transition diagram. Figure 13, a statechart, displays the two states that a LandUnit can have. The transitions come about because of events that are launched by internal activities (forestRecovering) or external actions (cut).

Activity diagram. The diagram shown in Figure 14 is the activity diagram of the “setWoodLotsProtection” method from the ForestDepartment class. The ForestDepartment sets protection for the WoodLots after comparing their size with the minimum authorized size. This threshold is updated through a request sent to the Farmers.

What is described in Figure 14 is somehow redundant with the details given in Figure 12. It is another way to represent the activity of the ForestDepartment.

Implementation of the “slash-and-burn” model in CORMAS

CORMAS overview

CORMAS provides a guide in building ABMs through its interface. It offers some facilities to incorporate data coming from geographic information systems (GIS) in order to define and describe “spatial entities.” Neighboring interactions among these “spatial entities” can represent natural dynamic processes (i.e., vegetation dynamics, erosion, pollutant diffusion); this is equivalent to a cellular automata layer. CORMAS also facilitates the design of “social entities” (the “agents”) representing the key stakeholders of the system under study. There is a set of predefined mechanisms for the location, perception, and movement of the agents, as well as for direct communication between them. Additionally, CORMAS has some tools to define specific markers (probes) to analyze simulation results, as well as viewpoints to allow visualization of the simulation from a particular perspective. It also provides a sensitivity analysis module to run sets of simulation experiments that automatically increases the values

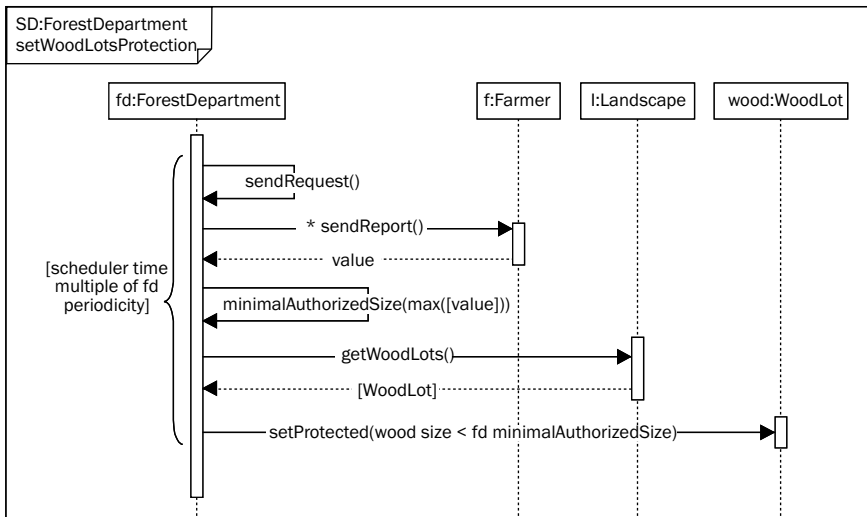


Fig. 12. UML sequence diagram of ForestDepartment's main step: setWoodLotsProtection.

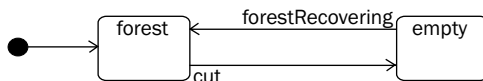


Fig. 13. UML state-transition diagram of the LandUnit class of the "slash-and-burn" model.

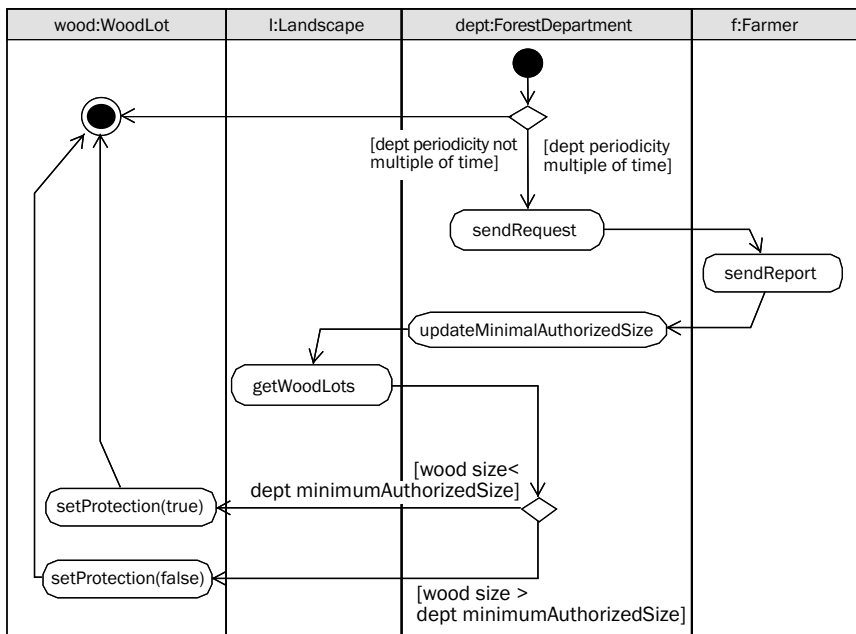


Fig. 14. Activity diagram of setWoodLotsProtection method from the ForestDepartment class of the "slash-and-burn" model.

of parameters within a given range. Finally, CORMAS allows exporting of the data produced by the simulation into spreadsheet or database software.

The CORMAS simulation toolkit is being developed continuously, through a step-by-step enriching process, by selecting what is of general interest in specific models and by “pushing it up” at the generic level. CORMAS comes with a library of existing models that can be divided into three categories: didactic models⁷ to illustrate the main concepts and principles of ABMs, theoretical models⁸ to investigate by simulation the field of theory building, and models oriented toward real-world case studies (Bousquet et al 2001) to better understand complex environments. Simple “quick and dirty” models, collectively designed with stakeholders through role-playing games, are also developed with CORMAS to support collective decision-making processes in complex situations (D’Aquino et al 2003). Stakeholders learn collectively by creating, modifying, and observing simulations. In such situations, CORMAS proved to be very convenient in allowing the integration of run-time modifications or new features suggested by the participants.

Adjusting the conceptual model to the CORMAS simulation platform

Starting from the class diagram of the model at the design stage (see Fig. 10), a new class diagram has to be designed to fit the particularities of the software that will be used. We thus adapt this first description of the slash-and-burn model to the CORMAS simulation framework. The idea is to use the generic CORMAS elements (classes with attributes and methods that already exist) as much as possible. A class diagram of the CORMAS Entity package is available at the CORMAS Web site.⁹ By taking advantage of inheritance from suitable generic spatial, social, and passive entities, some attributes and methods needed by the particular entities of the model are handled by reusing those existing at the more general level of the corresponding superclasses.

Figure 15 presents the class diagram of the “slash-and-burn” model adapted to fit the framework proposed by CORMAS.

The WoodLot class is set as a specialization of SpatialEntityAggregate. The LandUnit class is set as a subclass of SpatialEntityElement, which represents the smallest spatial entity (minimum granularity level) in CORMAS. To account for the concept of Landscape, we refer to the generic class called SpaceModel in CORMAS, which allows us to create and to refer to all the spatial entities. ForestDepartment is set as a kind of Agent, and Farmer is a specialization of AgentLocation, which is a kind of Agent located on a SpatialEntityElement in CORMAS.

A new class, denoted SlashAndBurn, now appears in the UML class diagram represented in Figure 15. This class is devoted to the design of simulation scenarios. In CORMAS, such a control level is specified through two roles: the first role is to create the initial situations for the simulation experiments, and the second role is to schedule the simulation experiments. The initialization process consists of creating and initializing all the instances from the classes corresponding to the conceptual entities of the model at time 0. Once created and initialized, these instances are stored in

⁷See, for instance, <http://cormas.cirad.fr/en/applica/plotsrental.htm>.

⁸See, for instance, <http://cormas.cirad.fr/en/applica/ecec.htm>.

⁹<http://cormas.cirad.fr/en/outil/uml-kernel.htm>.

collections that are automatically created as attributes of the SlashAndBurn class. In Figure 15, these collections appear as the roles called “theLandUnits,” “theWoodLots,” “theForestDepartments,” and “theFarmers.”

Now that we have a CORMAS superclass associated with each specific class of the SlashAndBurn model, not only attributes and methods of superclasses are directly reusable, but also associations between superclasses. For clarity, or just to emphasize important relationships, it may be convenient to “refine” such associations, which is a way to name them with a semantic adapted to the topic of the model. To signal this kind of refinement, we propose here to use one of the existing UML stereotypes: a symbol “/” that crosses the subassociation. For clarity, in Figure 14, we also shaded these associations dark gray. Hence, for instance, the aggregative association between the CORMAS classes SpatialEntityAggregate and SpatialEntity is refined to stress the importance of the aggregative association between the WoodLot and LandUnit classes.

Implementing the conceptual model within CORMAS

This paper does not discuss all the details of model implementation in CORMAS. The whole code of the model can be downloaded from the CORMAS Web site.¹⁰ To illustrate the translation of UML diagrams into the Smalltalk language that is used in CORMAS, we cite here the main method for the ForestDepartment class as an example. The corresponding UML activity diagram is shown in Figure 14. It is about the process of marking the WoodLots protected or not:

```
setWoodLotsProtection
```

```
“updates the Minimal Authorized Size and sets the ‘protected’ attribute of the woodlots to ‘true’
if its size is below this threshold”
```

```
((Cormas timeStep \ self periodicity) = 0) ifTrue: [
```

```
    self updateMinimalAuthorizedSize.
```

```
    self theWoodlots do: [:aWoodlot |
```

```
        aWoodlot protected: aWoodlot size < self minimalAuthorizedSize]]
```

To gain access to the WoodLots, the ForestDepartment requests the Landscape to perform the aggregation. In CORMAS, the SpaceModel class (which was used here to represent the concept of Landscape) is equipped with a set of generic aggregation methods. One of these generic methods (setAggregate:from:verifying:) is used here (see the code below):

```
theWoodLots
```

```
“Request the Landscape to perform the WoodLots aggregation”
```

```
“then get the updated collection of WoodLots”
```

```
self landscape
```

```
    setAggregates: WoodLot
```

```
    from: LandUnit
```

```
    verifying: [:c | c forest].
```

```
^self landscape spatialEntities at: #WoodLot
```

¹⁰<http://cormas.cirad.fr/en/applica/SlashAndBurn.htm>.

A simple simulation scenario

To be able to run a simulation experiment, a “scenario” has to be specified. With CORMAS, the scheduling process is based on discrete time-steps. In contrast to events-driven schedulers that activate the agents when given events occur, a discrete time-steps scheduler activates the existing instances of the model on a regular basis. In the SlashAndBurn model, the time-step duration is equivalent to one year. Each year, as is summarized in the sequence diagram shown in Figure 11, the scheduler activates the land units for forest recovering, then activates the farmers to perform their annual activities, and, finally, but only every 10 years, the ForestDepartment performs its activity. We decided arbitrarily to use the value 10 for the specific internal periodicity of activities of the ForestDepartment. This choice is denoted in the UML class diagram at the design stage (see in Figure 9 “periodicity” in the definition of the ForestDepartment class).

This is only half of what is called a “scenario.” We also need to define an initial situation; this means creating the desired number of entities, and assigning initial values to all the attributes of each entity. With CORMAS, it is possible to load the initial values of the attributes of elementary spatial entities directly from an ASCII file.

The other initial values are given directly in the UML class diagram at the design stage (see Fig. 9). One additional parameter is used to initialize the population of farmers: the total number of farmers, which here is a constant arbitrary number set at 40. This parameter could have been set as an attribute of a conceptual entity “Population,” but it does not make sense for this particular model. Because 40 is related only to the initial instantiation of the farmers, it is defined as a characteristic of the slash-and-burn model itself. The initial spatial distribution of the population of farmers also needs to be specified. In this case of simple simulation scenario, each farmer is randomly located on one of the 2,500 LandUnits.

Using markers (probes) to compare scenarios

Markers do not necessarily belong to the model itself unless they are used internally by any particular entity as criteria for a decision-making process. Markers may also be considered as external viewpoints established by anyone who examines the simulation with specific appraisal criteria. CORMAS provides facilities to employ such markers. The designer of the model has to write “probes,” which are Smalltalk methods that return the values that are automatically recorded by CORMAS at the end of each simulation time-step. The user may choose to export these data or to plot them as time-series within CORMAS. The three markers used in this case to compare scenarios are the number of forested LandUnits, the number of WoodLots, and the WoodLots’ mean size.

Measuring model sensitivity

To test the variability of the results when some randomness is incorporated into the model (random numbers are typically used to break ties among equivalent possibilities), it is necessary to repeat the same simulation experiment. To be able to perform a statistical analysis, a reasonable number of replications (at least 30) should be done.

The scenario builder of CORMAS proposes to select the parameters whose sensitivity is tested automatically. For each of these parameters, a range of values and a step of variation are given.

We propose here to test the probability of one farmer being conformist. Being equal to 0 means that all 40 farmers are behaving according to the nonconformist strategy; on the other hand, being equal to 1 means that all 40 farmers are behaving according to the conformist strategy. If we let this parameter range from 0 to 1, with a step of variation of 0.25, it defines five different simulation experiments; as it is to be repeated 30 times, this makes a total of 150 simulation runs.

Results

How many time-steps should we run in the model? This question is often crucial when some of the underlying assumptions become unrealistic in the long term. With this toy-model, we decided to run 300 time-steps for each simulation experiment, mainly because then the landscape evolution has converged toward a stabilized situation.

Rather than producing crudely the 30 time-series for the three markers for the five simulation experiments, we present here the average and standard deviation values calculated at the final time-step ($t = 300$) from the 30 repetitions (see Fig. 16). By doing this, we can discuss the effects of the proportion of conformist farmers in terms of final states, but not in terms of trajectories.

These results suggest the existence of an exponential relationship between the proportion of conformists and the number of forested LandUnits (Fig. 16A), and a sigmoid relationship between the proportion of conformists and the number of WoodLots (Fig. 16B). On the other hand, it seems impossible to detect any clear relationship between the proportion of conformists and the WoodLots' mean size (Fig. 16C), although, when the population of farmers is made up exclusively of conformists, the WoodLots are twice as big as when there are some nonconformists.

We will not discuss much here about the significance of such relationships. We can simply note that the first marker (number of forested land units) is somehow a combination of the two others (number of WoodLots and WoodLots' mean size). By just assessing the "ecological impact" of the farmers' strategy by looking at the number of forested land units, and/or by looking at the number of WoodLots, we can talk about a "gradual positive impact" of the proportion of conformist farmers. Actually, as soon as there are some nonconformist farmers in the population, the WoodLots' mean size does not increase.

Conclusions

We presented a model prototype and the main stages of its design—from a literal description of the context to a set of UML diagrams describing its structure and dynamics, up to its implementation and some simulation results. This highlights the development of the static model and its evolution from the "analysis stage" up to its adaptation into the CORMAS framework. Indeed, ABMs are often considered as black boxes containing hidden strange behaviors. Some simulation outputs may come from bugs or from biases that lower our confidence in the simulation results. To improve this situation, we emphasize three crucial points.

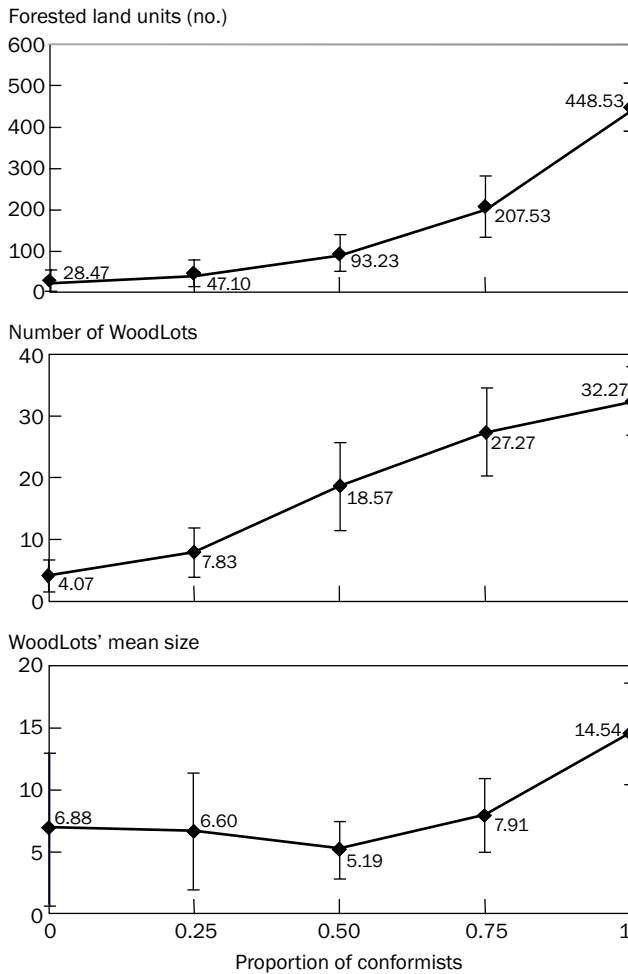


Fig. 16. Average and standard deviation values for (A) the number of forested land units, (B) the number of WoodLots, and (C) the WoodLots' mean size for increasing values of the proportion of conformists within the population of 40 farmers.

The model structure should be described from scratch, without reference to any simulation software. Description of a model should be sufficiently clear to implement it on any platform. A static or dynamic UML diagram should be as clear and simple as possible, and at the same time without missing crucial information. Because the UML formalisms contain only a few meaningful elements, notations should be strictly respected to obtain the essence of a model without ambiguity. The UML diagrams and textual documents should be considered as the “real” model; the model’s translation into computer code has to be seen as just one implementation. A single model description should be enough to get the same results when replications are run on various platforms. “A result that is reproduced many times by different

modelers, reimplemented on several platforms in different places should be more reliable,” according to Hales et al (2003). The benefit of designing conceptual models before rushing to implementation is not only a matter of enabling replicability. Following Heemskerk et al (2003), we believe that conceptual models are efficient tools to foster collaborative work between ecologists and social scientists. Modeling with UML does not mean that the model is well designed!

From an epistemological point of view, as stated by Popper, a UML model, like any other model, should be refutable (Popper 1985). Although never completely attaining formal proof, we can become more confident of a model over time by inductively analyzing the simulation results through sensitivity study. Designing and coding a model is only half of the work. Evaluating a model by means of sensitivity analysis is the other half of the modeling process. It may lead to modifications of the model, when new questions come up. This dynamic loop nurtures a learning process.

References

- Bergenti F, Poggi A. 2002. Supporting agent-oriented modelling with UML. *Int. J. Softw. Engin. Knowl. Engin.* 12(6):605-618.
- Bonnefoy JL, Le Page C, Rouchier J, Bousquet F. 2000. Modelling spatial practices and social representations of space using multi-agents. In: Ballot G, Weisbuch G, editors. *Application of simulation to social science*. Paris (France): Hermès. p 155-168.
- Bonnefoy JL, Bousquet F, Rouchier J. 2001. Modélisation d'une interaction individu, espace, société par les systèmes multi-agents: pâture en forêt virtuelle. *L'Espace Géograph.* 1-2001:13-25.
- Bousquet F, Bakam I, Proton H, Le Page C. 1998. CORMAS: common-pool resources and multi-agent systems. In: Pasqual del Pobil A, Mira J, Ali M, editors. *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Benicassim (Spain). Berlin (Germany): Springer-Verlag. p 826-837.
- Bousquet F, Le Page C, Bakam I, Takforyan A. 2001. Multiagent simulations of hunting wild meat in a village in eastern Cameroon. *Ecol. Mod.* 138:331-346.
- D'Aquino P, Le Page C, Bousquet F, Bah A. 2003. Using self-designed role-playing games and a multi-agent system to empower a local decision-making process for land use management: the SelfCormas experiment in Senegal. *J. Artif. Soc. Social Simul* 6(3). <http://jasss.soc.surrey.ac.uk/6/3/5.html>.
- Deadman PJ, Schlager E. 2002. Models of individual decision making in agent-based simulation of common-pool-resource management institutions. In: Gimblett HR, editor. *Integrating geographic information systems and agent-based modeling techniques for simulating social and ecological processes*. Santa Fe Institute Studies on the Sciences of Complexity, Oxford University Press. p 137-169.
- Ferber J. 1999. *Multi-agent systems: an introduction to distributed artificial intelligence*. Reading, Mass. (USA): Addison-Wesley. 509 p.
- Gilbert N, Bankes S. 2003. Platforms and methods for agent-based modelling. *Proc. Natl. Acad. Sci. USA* 99(3):7197-7198.
- Hales D, Rouchier J, Edmonds B. 2003. Model-to-model analysis. *J. Artif. Soc. Social Simul.* 6(4). <http://jasss.soc.surrey.ac.uk/6/4/5.html>.
- Hardin G. 1968. The tragedy of the commons. *Science* 162(1968):1243-1248.
- Heemskerk M, Wilson K, Pavao-Zuckerman M. 2003. Conceptual models as tools for communication across disciplines. *Conserv. Ecol.* 7(3):8. www.consecol.org/vol7/iss3/art8.

- Jager W, Janssen M, De Vries HJM, De Greef J, Vlek CAJ. 2000. Behaviour in commons dilemmas: Homo Economicus and Homo Psychologicus in an ecological-economic model. *Ecol. Econ.* 35(3):357-379.
- OMG. 2003a. Unified modeling language specification. March 2003 Version 1.5.
- OMG. 2003b. UML 2.0 Superstructure specification. Final adopted specification. August 2003.
- Popper KR. 1985. *Conjectures et réfutations: la croissance du savoir scientifique*. Paris (France): Payot.

Notes

Authors' address: 467 593 827. CIRAD – TERA, TA 60/15, 34398 Montpellier Cedex 5, France, {bommel}{le_page}@cirad.fr, phone: +33 467 593 839, fax: +33 467 593 827.