

---

## Description Formalisms in Agent Models

---

### 2.1. Introduction

This chapter will aim to present good practice in, and the benefits of, formalization in modeling multiagent systems (MAS). To achieve this, the authors will first reiterate the usefulness of modeling systems, while placing the paradigms associated with a multiagent approach in context. Then, they will argue that the use of graphic modeling languages enhances the exchanges between the parties involved in the design of an MAS. Following this, two types of graphic models based on the same semantic base are presented: *Unified Modeling Language* (UML) and *Agent Modeling Language* (AML). The first graphic model is intended for general use and facilitates its users to analyze the ontology and dynamics of the modeled system. The second graphic model uses paradigms specific to agents and facilitates its users to create a design which is closer to the MAS which will be produced. After having discussed the relative merits of each of these graphic model types and presented some possible extensions, the chapter discusses the utility of, and a method for, documenting a multiagent model. In order to do this, the *Overview*, *Design concepts*, *Details*

(ODD) protocol, which guides the modeler in the creation of a documentation of the objectives, constitutive elements and specific properties of the model, is presented.

We illustrate each of the concepts presented (UML, AML and ODD) through their application to an example which will be a recurrent theme in the remaining of this chapter.

## **2.2. Recurrent example**

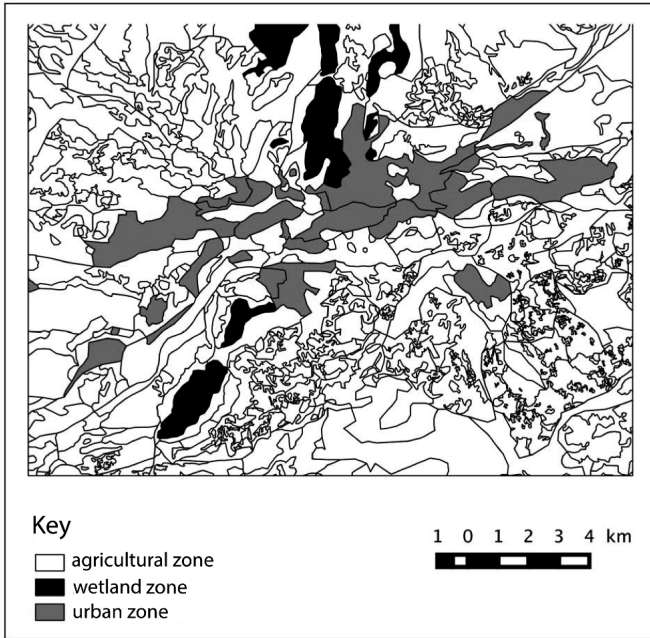
Many applications exist which have clearly demonstrated the utility of the agent approach for modeling complex phenomena. These involve numerous domains, such as ecology, social science or epidemiology. We have chosen to address the domain of epidemiology because, first, this theme takes in several other domains, in particular ecology and social sciences, and second, a wide range of multiagent concepts can be involved in modeling complex phenomena such as these.

As such, this chapter, and those that follow it, will be thematically linked through a recurrent modeling example which is based on an epidemiological phenomenon. The phenomenon in question is the geographic dispersion of an epidemic transmitted to humans by mosquitoes. The aim of the model is to understand and measure the impact of the pendular journeys people take (moving from home  $\leftrightarrow$  work) on the development and spread of a contagious disease.

We have decided to study malaria, which is present in many African countries, and we apply this to the Maroua subregion of Cameroon (see Figure 2.1).

The only way in which malaria is transmitted to humans is through the bite of an infected anopheles mosquito. A healthy mosquito becomes infected when it bites an infected person. People cannot pass the disease on to other people, and mosquitoes cannot pass the disease on to other mosquitoes. By itself, the mosquito has a very small movement radius, of approximately 50 m a day, but it is present throughout the territory.

However, the real vector for the spread of the disease seems to be people because they need to travel over large distances to conduct their daily activities.



**Figure 2.1.** *Map of the Maroua subregion (Cameroon)*

Most of the inhabitants of the Maroua live in urbanized areas (black in Figure 2.1). The crop farmers of the region travel in a pendular movement between their homes (in town) and the area that they farm, outside the town (white and gray areas in Figure 2.1). The wetland zones correspond to wet areas where the mosquitoes can lay their eggs.

In the following, this system will be modeled using UML (section 2.3.1), and then AML (section 2.3.2), and will be documented using the ODD protocol (section 2.4). In this way, the readers will discover the different forms of model description through an example of each.

### 2.3. Formalization of agent models

In this section, we will first detail the reasons for formalizing MAS, and then we will present two tools designed to aid this formalization – UML and AML – which we will illustrate using our recurring example.

Formalization is a process which has three goals. The first goal is related to the system under study. A tool is required that is suitable for gaining an understanding of the system concerned. The second goal is connected with abstraction, which helps us to not be restricted by technical considerations linked to the simulation components. The third goal is the generation of a code, which will make it possible to transition from model to implementation. In addition, graphic formalizations, such as UML and AML, also allow us to streamline communication about the content of the model among several people. These languages have the advantage of being able to synthetically show, using one or several diagrams, complex mechanisms and structures. The graphic nature of these diagrams simplifies their understanding for non-programmers.

In this chapter, we have chosen to concentrate on UML and AML; the latter is an extension of the former, and it is more specifically dedicated to the agent paradigm through its formalization of the description of agent behaviors and interactions.

The various diagrams developed from UML may also, of course, be used for model development. However, its generic nature causes certain problems when it comes to using it within a specific context. Thus, for spatialized simulations, it is necessary to use a language which can represent space and its constraints, such as the representations underlying Geographic Information Systems (GIS) [CHI 13].

#### 2.3.1. *UML*

In the 1970s and 1980s, there was disagreement between those who believed in modeling data and those who believed in functional modeling. In this period, the use of flow and relational diagrams was generally considered to be mutually exclusive.

These two camps finally came to an agreement at around the end of the 1980s, and realized that most projects could benefit from the use of both model types. This reconciliation was followed by the emergence of numerous object-oriented analysis and modeling methods. However, each method had its own specific notation and definition of terms such as object, type and class. There was no common standard. The number of modeling languages increased from less than 10 to more than 50 between 1989 and 1994.

At the end of 1994, Grady Booch and Jim Rumbaugh announced their collaboration on the development of a *Unified Method*. They were later joined by Ivor Jacobson. In the end, after several years of experimentation with various notations and concepts, the group established a semantics for object-oriented concepts and agreed on a common notation on the basis of several of the notations and concepts with which they had experimented.

During 1996, the *Unified Method* developed into the UML. This new name was designed to emphasize the fact that UML was a modeling language and not a method. Its aim was to provide an expressive notation to define a semantics for implied concepts, and to leave the development process choice open. In the end, UML, developed from the combination of the three methods of object modeling, Object Modeling Technique (OMT), Booch and Object Oriented Software Engineering (OOSE), became an essential standard. Originally created to enable a developer to represent, specify, analyze and visualize the structure of a project in object-oriented programming, UML is today used in a large number of fields.

The development of UML is quite like the development of software in that there are *major* versions as well as improvements and extensions. The current version of UML is version 2.0 [OMG 05], which is divided into four parts:

- *UML 2.0 Superstructure*: diagrams used for modeling;
- *UML 2.0 Infrastructure*: foundations shared with MOF 2.0;
- *UML 2.0 OCL*: the language of constraints;

– *UML 2.0 Diagram Interchange*: makes it possible for exchange of diagrams between tools to take place (including from a graphic point of view).

UML Superstructure contains various types of diagrams:

- *six structure diagrams*: classes, objects, composite structures, components, deployments and packages;
- *three behavior diagrams*: activities, use cases and state machines;
- *four interaction diagrams*: sequence, communication, overview of interactions and timing.

#### 2.3.1.1.1. *Formalization of model structure (static diagrams)*

This section presents the descriptive aspect of UML modeling. This is also known as the static part, or the structure. Here, we will only present two of the six structure diagrams: class diagram and object diagram. These are the diagrams that are most widely used in UML modeling. The class diagram is also used for meta-models (models of models), also known as ontology.

##### 2.3.1.1.1.1. Class diagram

The UML class diagram allows us to model the structure, i.e. the static part of a system. Classes are essential in that they define an abstract type which will later make it possible to instance objects in the object diagram. Figure 2.2 presents the complete class diagram for our recurring example. We can see the class Entity. This class possesses the Boolean-type attribute *is-infected* and a function *infect()* which makes it possible to infect another Entity. Furthermore, this is a situated entity, and thus possesses the attribute *CurrentPosition* which is the location in which it is found, and the method *move*. In this diagram, classes are not isolated. There may be links between them, known as relations. There may also be relations between classes for various reasons:

- *Heritage* allows one class to inherit all of the attributes and methods of the mother class from which it is descended. This relation is represented by a bold arrow. Thus, the classes Mosquito and Human inherit the attribute *is-infected* from the class Entity. In

addition to attributes and inherited methods, the class `Mosquito` also possesses the attribute `patient0`. This represents the mosquitoes that are contaminated at the initialization of the simulation, known as patient zero. This patient is represented by a Boolean which is true in this case and would otherwise be false.

– *Association* makes it possible to link two classes. It may be named and may contain information on multiplicity (cardinality) and navigation (direction of the relation). In the example, we have shown the association `Has` contaminated which is connected in a specific way, because it links `Entity` and itself (reflexive association) to indicate the chain of infection, showing who has contaminated and the infected entity. The *cardinality* “\*” indicates that an entity can be the contaminator of zero or several entities, and the cardinality “0..1” at the other end of the association indicates that an entity has been infected by zero or a single contaminator. There are two special association cases: *aggregation* and *composition*. These are represented, respectively, by an empty and a filled-in diamond on the aggregate side. In our example, a composition relationship links the places (the class `Place`) to their `Territory` (aggregate).

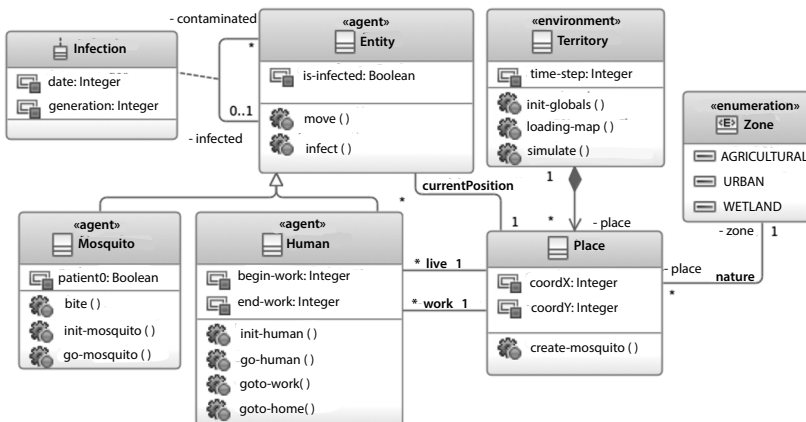


Figure 2.2. Representation of the class diagram from the recurring example

#### 2.3.1.1.2. Representation of the recurring example with UML

In the UML model, we have grouped the common elements of the mobile entities (mosquitoes and humans) together in a class `Entity`. We represent the concept of contamination between entities with an association which has attributes, called “Infection”. The information does not appear in the diagram (because we have not presented the constraints), but an entity cannot be infected by an entity of a different subtype. For this reason, information regarding the source is maintained through association, and information concerning the date and generation is also maintained through the association attributes.

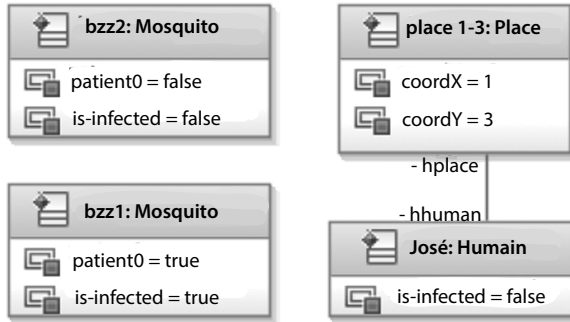
The territory is made up of elements from the class `Place` which corresponds to the various possible soil occupation zones, specified through the attribute “nature” in the class `Place`. This is an enumerated-type attribute, whose various modalities are specified in the class `Zone`. Furthermore, in our representation, the `Territory` also plays the role of environment for the system and contains the representation of time (time-step) in the simulation and the entire management part of the system. We find these same elements of simulation management for beginning the simulation after the various components have been initialized and the card has been charged. A `Place` is denoted by its coordinates, which represent the center of the zone. Initially, it needs to be possible to create infected mosquitoes. This is performed by the method `create-mosquito`.

#### 2.3.1.1.3. Object diagram

The object diagram facilitates the classes defined in the class diagram to be instantiated as real objects. This diagram is useful for giving an image of the state of the system at time  $t$ . In order to model the initial state of the system, for example, an object diagram is used. In Figure 2.3, we have represented three entities, two of which are mosquitoes (a zero patient and an uncontaminated mosquito) and the last one is a human. At time  $t$ , the communication chain is limited to the zero patient. It is possible to give object values to all or part of the attributes. In the example, we have not shown all of the objects and relations. For example, the agents are situated and they should be in a



relation with the instances of the class `Place`. Furthermore, we have only represented one `Place1-3`.



**Figure 2.3.** *UML object diagram*

#### 2.3.1.1.4. Meta-model

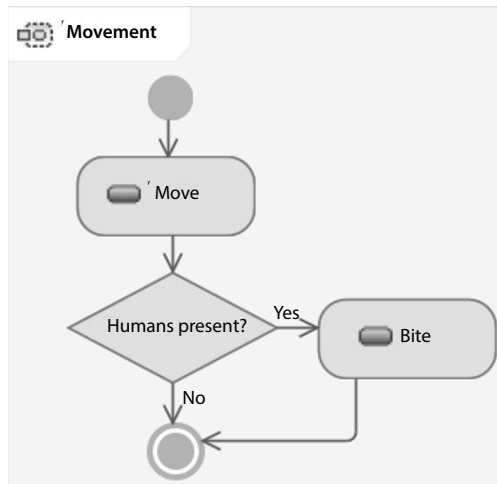
A meta-model is a modeling language which makes it possible to describe another language, much like grammar which is used to describe real language. In UML, the class diagram represents all of the elements permitting the description of a UML model. This means that the language can represent or define itself by itself, and can also define a new framework for modeling. In this way, it becomes possible to extend or specialize UML, as suggested by the creators of AML. We will observe this in much detail in the section that follows.

#### 2.3.1.2. *Formalization of model operation (dynamic diagrams)*

This section presents the analytical diagrams, which are also sometimes known as dynamic diagrams, because they enable the description of the dynamic aspect of the system. It brings together all of the behavior diagrams and the interaction diagrams. Of the seven possible diagrams, we will only present three diagrams here: activity diagram, state-transition diagram and sequence diagram. We will conclude by discussing the coherence verification features. Coherence verification can be carried out on the basis of meta-modeling elements, and using all of the information provided by the various diagrams.

### 2.3.1.2.1. Activity diagram

The UML activity diagram is one of the diagrams which allow the modeler to represent the behavior of an object using nodes (of activity, action, control or objects) and transitions. Activity diagrams are suitable for specifying sequential or concurrent treatments. They provide an overview of the control flows from one activity to the other. In Figure 2.4, we show activity linked to the movement of a mosquito. This can be seen as an activity related to the method `go-mosquito`. Thus, the mosquito moves about and, if there is a human in the area, it bites them.

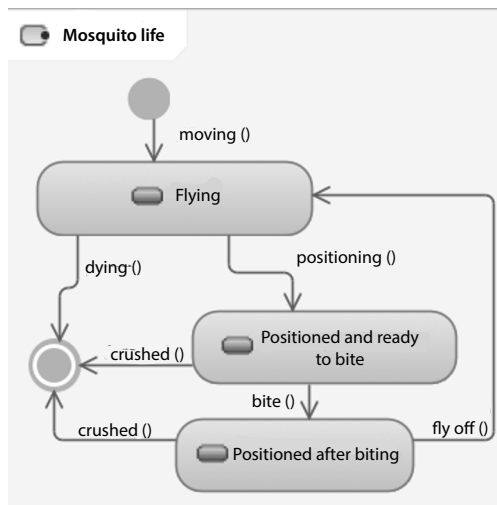


**Figure 2.4.** *UML activity diagram*

### 2.3.1.2.2. State-transition diagram

The role of the state-transition diagram is to represent finite-state automata (i.e. entities that are characterized by a set of states which, at any given moment, are in a specific state) in the form of a set of transitions, which may or may not be labeled. A state is characterized by the value of the attributes of a system at a time  $t$ . A transition represents the transition from one state to another; such a transition is generally triggered by an event. This triggering may be automatic, when the event that triggers the change is unspecified. It is also possible to condition the

triggering of a transition using *guards*: these are Boolean expressions, expressed in natural language or in *Object Constraint Language (OCL)*, for example. In Figure 2.5, we show the lifecycle of a mosquito. It begins its life and moves around until it dies or is killed by a human. If this does not happen, it lands on a person and biting them each time is possible. During the exchange of fluid, one or the other of the entities involved may become infected with malaria. It should be noted that the mosquito never rests and as soon as it is in the same place as a human, it will bite the human.



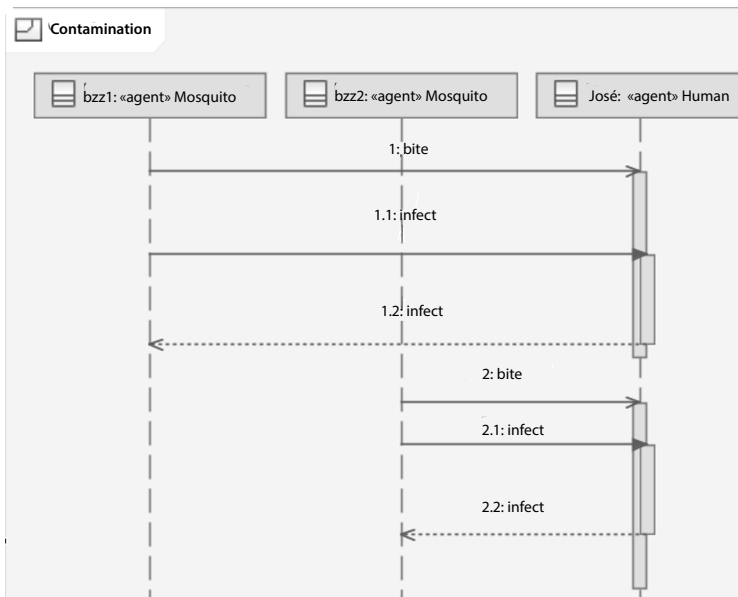
**Figure 2.5.** UML state-transition diagram

### 2.3.1.2.3. Sequence diagram

Sequence diagrams are used to represent interactions between the modeled system's entities (actors or objects). It makes exchanges (synchronized or non-synchronized) visible. There is a composition language available, which means that parallel treatments can be expressed using interaction frameworks which may contain algorithms. In Figure 2.6, an exchange is shown between the three entities of the object diagram, showing a contamination cycle.

#### 2.3.1.2.4. Consistency checking

As with any representation, it is important to check the consistency of the model. Currently, there are tools available which can help make these checks, such as ATL. ATL<sup>1</sup> is a model transformation language, which works at the meta-model level, and has been in development since 2003 at the University of Nantes. It aims to make it possible to express model transformation rules and to execute them. Since January 2007, ATL has been part of the Eclipse *Model-to-Model* (M2M) section, and thus it is recommended for use as a tool for transforming one model into another. In addition, it is integrated as a *plugin* to the design platform Eclipse. To achieve this, a meta-model needs to be defined, which enables the representation of a model, as shown in Figure 2.7.

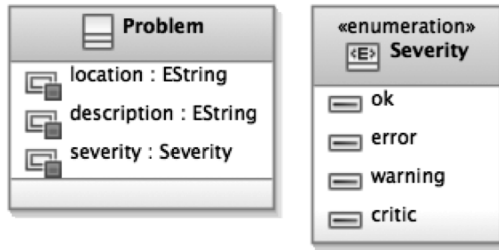


**Figure 2.6.** UML sequence diagram

This meta-model contains a meta-class *Problem* and a meta-list *Severity*. This meta-model allows us to instance problems detected

<sup>1</sup> ATL is an abbreviation of *ATLAS Transformation Language*.

in the model in order to identify their source (attribute `location`), description and severity. Their severity lets us know whether the error is critical for code generation. If this is the case, then no code is generated. On the other hand, if there are only warnings and not critical errors detected, then code generation takes place.



**Figure 2.7.** *Problem meta-model*

Modeling like this can be carried out using standard UML; however, when it is conducted in this way, it would be initially difficult to understand and could not be used as a tool for communication.

This is why the following section discusses the AML extension which has been added to UML.

### 2.3.2. AML

Since the UML language is very general and is oriented toward object programming, much work has been conducted on the development of a suitable graphic formalism, which might perhaps even be dedicated to the agent paradigm.

The development, since the mid-1990s, of MAS design methods (e.g. Gaia [ZAM 03], TROPOS [BRE 04] or Prometheus [PAD 02]) has naturally been accompanied by the development of various dedicated graphic modeling languages (e.g. Agent UML (AUML) [BAU 01b] or AML [CER 07]) which are more or less based on UML, and which aim to supplement UML with concepts specific to the domain of MAS. Of these languages, we will be particularly concerned with

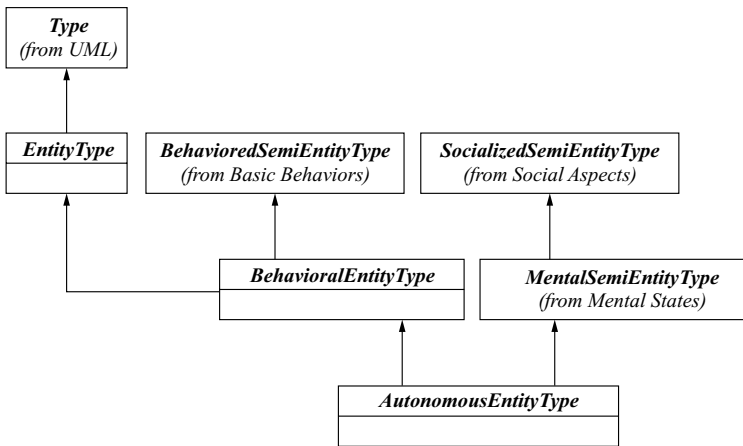
AML [TRE 05, CER 07]. AML is a semi-formal language for modeling and documenting applications based on MAS. It was developed as an extension to UML 2.0 [OMG 03]. The main aims of the designers of this language are to include and bring together the preexisting concepts from various agent architectures (in particular, *Beliefs Desires Intentions* (BDIs)), and preexisting languages and modeling methods. Thus, AML seeks to be suitable for any type of agent-based applications and to be independent from the methodology and development platform used and from the case to which these are being applied.

### 2.3.2.1. *Formalizing model structure (agent-group-role)*

#### 2.3.2.1.1. Agents

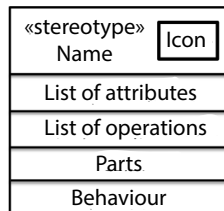
This section emphasizes the description of the model structure, which in UML is conducted mainly by using class diagrams. In contrast to UML, with which only classes may be defined, by using AML we can refine and represent various types of entities involved in MAS and agent-based simulations. Figure 2.8 is a fragment of a complete AML meta-model. It presents the hierarchy of AML entities on a general level. A more detailed description will not be given here. Similarly, the concepts and formalisms used in the structure model are lower level concepts. For example, we will directly use the concept of agent without reference to the fact that it is a specialization of *AutonomousEntityType* which is descended from the entities *BehavoredSemiEntityType*, *MentalSemiEntityType*, etc.

The principal entities are the agents (autonomous entities which can perceive, interact and have a certain behavior within their environment; these also have mental attitudes and social abilities), the resources (entities without their own autonomous behavior, which are used by agents; their availability is an important characteristic) and the environments (the entities within which the other entities develop). AML also makes it possible to define the entities' role and organization, which are linked to the social capacities of the agents. These two concepts are close to those presented in the Agent, Group, Role (AGR) model [FER 04]. An organization also means that a set of agents which are considered at a higher level as a single agent can be represented.



**Figure 2.8.** Entity hierarchy in AML






Each entity is represented, graphically speaking, by a UML class with a stereotype unique to the entity and/or an icon, which specifies to which type the entity belongs. Figure 2.9 depicts the entity. It is characterized by a list of attributes, a list of operations, the elements which composes it (the field part in UML 2), and behaviors. The *parts* are particularly useful for specifying which agents make up an organization. Behaviors are complex actions composed of operations or perhaps even other behaviors. The way in which these are conjugated for different types of entities is shown in Figure 2.10.



**Figure 2.9.** Description of an entity in AML

Finally, the AML formalism is very open and it is possible to describe the same element of the system within the modeled system in several ways. Thus, in the description of the formalism that follows,

we will detail the methodological choices that we have made. We will present AML modeling based on the AGR approach.

Type of entity	Stereotype	Icon
Agent	«agent»	
Environment	«environment»	
Resources	«resources»	
Organization	«organization unit»	
Role	«entity role»	

**Figure 2.10.** Stereotype and icon for each type of entity

The agent is the central element of the modeling. At the lowest level, it is described as in Figure 2.9. At higher levels, it is possible to refer to an agent that has already been described, simply by using its name and the *agent* icon.

An agent is identified by a name, and for a given agent there may be several instances, and thus several agents of the same *type*. It is, therefore, important to make a distinction between the agent *type* (which is close to the UML class concept) and the agent itself, which will be present within the system (instance).

As shown in Figure 2.9, following this we find the list of agent attributes and the list of its operations (the actions that it can carry out). The *parts* part of the agent will always remain empty within this chapter. In fact, for the purposes of this work, we consider that an agent cannot be made up of other agents (as is the case in an implementation in NetLogo)<sup>2</sup>. The final part, *behaviors*, contains the *fragment behaviors*. These make it possible to describe the *complex* behaviors that the agent

<sup>2</sup> In contrast to NetLogo, other agent-based modeling and simulation platforms (such as GAMA [GRI 13] or Corman [LEP 12]) allow for the definition of agents as being made up of other agents.



can accomplish by breaking down them into several simpler actions. *Fragment behaviors* can also be used to describe reusable actions by other agents. As described in section 2.3.2.2.2, this part of the agent is fulfilled only if the agent is able to conduct complex actions. This concept will be described in much detail in the interaction model.

#### 2.3.2.1.2. Groups

The concept of group does not exist as such in AML. In its place are *organization units*. These are used in AML to describe organizational structures, environments which specify social arrangements between the entities in terms of interaction, roles, constraints, etc. This approach is relatively close to the notion of group as it is introduced into the AGR model. As a result, we have decided to use *OrganizationUnitType* to describe the groups of our MAS. In the same way as for agents, it is essential to make a distinction between the group as a structure and the group as an instance.

In the AGR approach, the group is an abstract notion, allowing us to bring together the agents which share characteristics or resources. As it stands, the AML formalism shown in Figure 2.9 is too rich. In fact, a group, as defined in the AGR model, has neither attributes nor methods. This formalism nonetheless allows us to model the notion of an agentified group.

Our model must actually be able to describe the fact that a set of agents may be considered as a unique entity at a higher level of abstraction. We represent this in AML using *OrganizationUnitTypes* and a specific role of *leader* for each group. This role will always have the name of leader followed by the name of the group to which it is attached.

In a model, a group will have its empty *attribute list*, *operation list* and *behavior* parts. The *parts* part is the only one that will be full: this is the part that makes it possible to describe the group's structure, that is to describe what it is made up of. A group may be made up of agents, other groups or possibly both. In this part the various roles which the group's agent may play are also found, with the specific role of *Leader* which is present in each group.

### 2.3.2.1.3. Roles

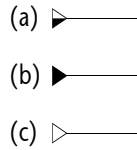
According to the AGR model, a role belongs to a group. In our structure model, we represent this by placing all of a group's roles in the group's *parts* part.

In AML, a role is described as shown in Figure 2.9. The concept of role is as defined by Ferber [FER 98] for the AGR mode. Thus, a role cannot be made up of other roles, and for this reason, the *parts* part is always empty. The three other parts may or may not be empty, depending on whether the role gives access to attributes, simple methods or complex methods.

The systematic existence of the specific *Leader* role for each agentified group should be noted. For this role, the *attribute list* part (respectively, *operation list*) allows us to describe the list of attributes (respectively, methods) of the agentified group, that is the agent playing the role of the group leader. The *behaviors* part makes it possible to describe the complex actions that the agentified group (and thus the agent playing the role of *Leader*) is capable of carrying out. As with a more classical role, these parts may be empty.

According to the definition of role in the AGR model, two more things need to be modeled: the fact that some roles require prerequisites to be carried out, and that some roles may make it possible to direct other agents. As AML is based on UML 2 and is compatible with it, the prerequisites may be described using OCL constraints on the relation of "*playing a role*"; the graphic representation of this is shown in Figure 2.12. The hierarchy between the roles is expressed using links between the roles, as shown in Figure 2.11.

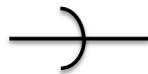
There are two types of relationships between entities: "peer-to-peer" relationships and "master/slave" relationships. There are three types of links between two roles in order to show these types of relationships. "Peer-to-peer" (*a*) relationships are shown by black and white triangles. These "peer-to-peer" relationships are the relationships between entities of the same social status and that have the same level of authority.



**Figure 2.11.** *Types of link between roles in AML*

Relationships of the “master/slave” type are shown by the links (b) and (c). The link (b), a black triangle, represents the leader (or *superOrdered* in AML). A leader can control the behavior of the agents under its command. On the other side of the relationship is the link (c), made up of a white triangle, which represents the subordinate. An (a)-type link is always associated with another (a) link, and the (b)-type link is always associated with a (c)-type link.

Now that we can link the roles with each other, let us see how we can link the roles to agents.



**Figure 2.12.** *Playing a role in AML*

Figure 2.12 shows the association between a role and an agent, which means that the agent plays the role in question. The half-circle faces toward the role. We can also specify the role’s multiplicity, that is the maximum number of agents which can play this role in this group at the same time.

We have now shown the three essential components of the AGR model, which are the agents, the groups and the roles. There are, however, two further entities which are used in the structure model: environment and resources.

#### 2.3.2.1.4. Environment and resources

The environment is a logical and physical entity which surrounds all the system's entities. Its AML representation is shown in Figure 2.9. There is a maximum of one unique environment in the structure model. The environment may not be present if it does not contribute anything to the model.

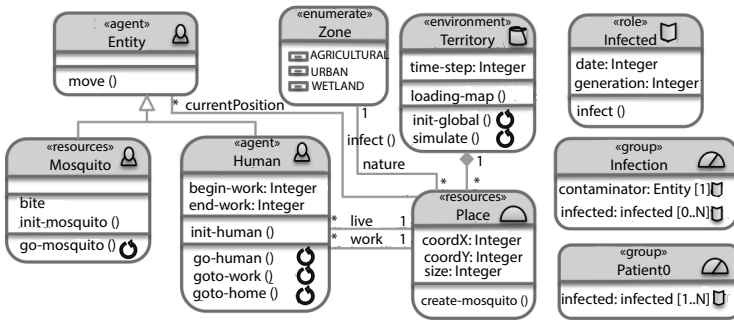
We have assumed that the resources are connected to the environment, as this is the global entity in our system. Thus, the *parts* part, renamed *Resource List*, contains the system's resource(s). The three other parts of the environment may be used. From the environment, we only consider the system's global variables (such as temperature) and the operations which make it possible to modify these.

The resources are physical entities (for example, raw materials in a production system) or computerized entities (for example, a database). *Simple* resources are accounted for in our system: these are not made up of other elements. As such, a resource will never have any *parts* in its representation (see Figure 2.9). A resource may possess one or several attributes (such as a capacity). Finally, in the case of an information technology (IT) resource, it is considered that this may contain simple or complex operations. In the case of a knowledge bank, for example, these operations may be used by agents which have access to it.

#### 2.3.2.1.5. Representation of the recurrent example with AML

In the AML representation of the recurrent example, we have chosen to take the opposite of the UML representation for the formalization of the link between infections. In fact, because AML allows us to adopt the AGR approach, we have decided to use this to represent the infection. Here, we have two roles. The first role is concerned with the contaminator. This is a specific role within a group, which allows us to find out the source of the contamination for all of the infections of the group *Infection*. The second role is concerned with the infected entity. It indicates, when an entity is infected, the date of the infection and where it came from. It also makes it possible to acquire the infection capacity (operation `infect()`) and for it to become, in turn, a contaminator of the group *Infection*. The second group `patient0`

makes it possible to manage mosquitoes that are infected at the system's initial state. As for the rest of the system, the same elements are present as with the UML model. One small variation is that complex behaviors based on other operations or system behavior, such as `goto-work`, which uses `movement` to go from a specific `Place` to another, from home to work, can be highlighted (using the circular arrow).



**Figure 2.13.** AGR representation with AML for the recurrent example

This concludes the descriptive part; let us now focus on the analytical part, which contains the interactions and the actions.

### 2.3.2.2. Formalization of model operation (interactions and actions)

In this section, we will present the modeling elements which allow us to describe the dynamic part of the models, in particular interactions and actions.

#### 2.3.2.2.1. Interaction model

There are two types of interaction: interactions between an agent and one or several other agents, and interactions between an agent and its environment, which is where the agent and its resources are based. Here, we will consider several interactions, which can be classified into five major types:

- *communication*: three types of communication may be listed: sending messages, using a blackboard and communication with markers (such as pheromones). A message may also have several consequences

for the agent: it may change its state, it may make it send a reply or it may make it carry out a particular action;

- *reaction*: a distinction is made between actions and agents, and the effects that these actions produce. An action carried out may have consequences on one or on several agents, as well as on the environment;

- *cooperative and/or complex actions*: some actions cannot be carried out by one agent alone, and require cooperation between several agents. Also, some complex actions may be broken down into several simpler actions. This is especially the case for the actions known as *at a high level of abstraction*, which will be discussed later in more detail;

- *scheduling*: due to the action model, which provides the starting and ending dates for each of the actions carried out, we can check if a scheduling is possible or not. Additionally, we can produce a naive scheduling if we are lacking this. In this way, it is possible to check that an agent does not complete an action before one which precedes it has been completed (by it or another agent);

- *knowledge and learning*: agents are able to memorize, learn and modify their basic knowledge as the system develops. As this part is in itself a very important subject, these two concepts are very minimal in our model, and their further development is one of the possible ways in which the system may develop.

#### 2.3.2.2.2. Actions


The AML formalism is open with regard to modeling choices, and thus the same model, even a very simple one, may be described in several ways with this formalism. We will, therefore, specify and detail certain modeling choices.

It is possible to describe the actions and capacities of agents on different levels. Therefore, the choice of modeling is made on the basis of the type of action. The discriminating factor is the fact that there is no *special* precondition or postcondition. We consider a condition to be special if it does not relate uniquely to the capacity and knowledge of the agent regarding the accomplishment of the action.

Thus, a simple action is an action without any special preconditions or postconditions. It will be described directly in the organizational part of the agent.

#### 2.3.2.2.3. Complex actions

Cooperative actions and complex actions are described in the structure model using the AML concept *fragmentBehavior*.

<<behavior fragment>> 
<b>Name</b>
<i>attribute list</i>
<i>operation list</i>
<i>parts</i>
<i>behaviors</i>

**Figure 2.14.** Description of a behavior in AML

Figure 2.14 shows what is known in AML as *fragment of behavior*. These are used in two cases: for describing behaviors which could be reused and for breaking down a complex behavior into several simpler behaviors. In the first case, this means that we make several behaviors correspond to a fragment of behavior. In this way, an agent which has this fragment of behavior has these operations. We do not use them as they are, but rather we use them to describe the actions that may be described as *complex*. Also, the parts *attribute list* and *parts* always remain empty. If the complex action may be divided into several simple actions, these will appear in the part *operation list*. If the complex action may be divided into several actions which are also complex in themselves, then these will appear in the part *behaviors*.

#### 2.3.3. UML versus AML

UML is very widely used. Its expressiveness means that it is a modeling and communication system which is suitable for our needs.

It can be used alongside other languages and/or formalisms, or can also have extensions added to it as necessary. OCL also allows us to express the properties of the system.

The UML approach in an MAS implementation can lead to additional complexity. In particular, this occurs when the model needs to include those MAS paradigms that it does not allow to be suggested natively (such as AGR). In this case, it is necessary to redefine these or to add information to link these with the implementations.

The risk that this approach leads to is the loss of illustrative and intuitive aspects. Thus, the complexity is not due to the system under study, but rather it is due to the modeling elements.

For example, how can the different levels of abstraction of which a system is comprised of be expressed? How can the autonomy of entities at each of these levels be expressed, with perception, knowledge, capacities and access to various resources?

Obviously, AML formalism, which has been defined for this very purpose, does offer these concepts. Those who do not like it argue that it is unfortunately not widespread. This comes partially from the fact that there are only a few tools which support the use of this language.

It is important to note that AML is extremely powerful and modular; this is in part due to the fact that it is descended from UML. However, AML is not easy to grasp as a whole. In addition, all of its complexity is not necessarily useful in the context of agent-based simulation. In this chapter, we have not presented an exhaustive list of all the concepts which exist in AML, but rather we have limited ourselves to those that seem relevant to the domain of agent-based simulation, and which have been used in the example. For a more complete overview of the concepts available in AML, the readers may refer to the reference work on AML [CER 07].

In this chapter, we will be concerned with the link between the models, in particular between UML and the tool NetLogo.



#### 2.3.4. *Other variations of UML*

Although there are other ways of modeling MAS, these methods are for the most part focused on a particular field, and cannot be used in a more general case. However, we should highlight several of the works which propose a modeling or specification language: Agent Communication Language specification from the Fondation for Intelligent Physical Agent (FIPA ACL) [FOU 97], Knowledge Query and Manipulation Language (KQML) [FIN 94], Taming Agents and Objects (TAO) [SIL 03], Object-Process Methodology for Multi-Agent System (OPM/MAS) [STU 03], AUML [ODE 00, BAU 01a] and [ODE 01].

The most developed and successful of these approaches is the AUML approach. AUML [ODE 00, BAU 01a] suggests mechanisms for modeling interaction protocols in MAS. In order to achieve this, a new diagram class has been introduced in UML: protocol diagrams. These extend UML state and sequence diagrams in several ways. The specific extensions introduce the agents' roles and their simultaneous execution sequences, extend the messages' semantics and model the interaction protocols.

AUML was put forward and accepted for inclusion in the standard FIPA'99. However, AUML does not offer a graphic solution to the problem of agentified groups.

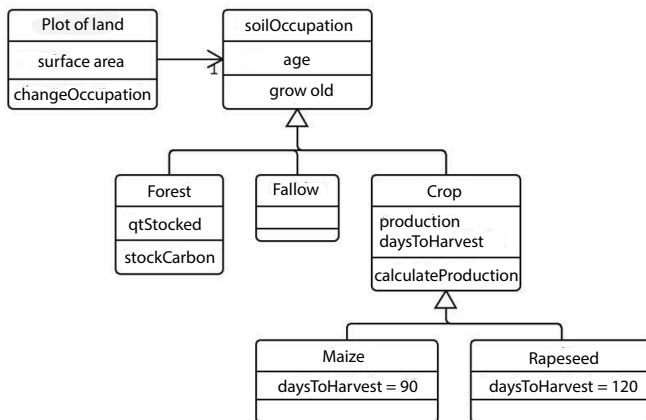
#### 2.3.5. *Formalizing changes in behavior in UML*

The concept of role which we outlined in the AGR approach is also useful for specifying a behavior which will change over time at the level of the same entity. Although this concept is directly available in AML, it is not when the modeler decides to use a UML formalization. In this section, we will show how modelers can structure their model to show changes in behavior; this is achieved through the use of a *Design Pattern* such as Actor-Role [COA 97].

The Actor-Role design pattern involves representing the attributes and actions linked to a behavior within a specific class (the class role),

and which is associated with the Agent class. As Bommel explains: “when we concern ourselves with the representation of humans or animals which can evolve, transform, and change behavior throughout their life, it is useful [...] to associate a specific behavior with the agent that plays this role, [...] in order to allow it to change role (and thus behavior) over time” [BOM 09]. In this way, when an entity changes behavior, no major modification needs to be done to the entity, and all that needs to be done is to associate a new behavior to it. When this logic is applied to spatial entities, we more readily refer to state rather than role. The application of this pattern thus consists of specifying a distinctive class to which the spatial entity pertains, along with the actions and attributes linked to a certain state. This structure proves itself to be particularly relevant in the modeling of occupations of space, to which states bring specific dynamics. It also makes it possible to organize these dynamics clearly, and thus to communicate more effectively with regard to the content of the model.

We propose to extend our recurrent example to present the application of the Actor-Role pattern to the dynamics of spatial entities. Thus, we will illustrate its use on a well-known case in the study of change in soil occupation.



**Figure 2.15.** A class diagram for modeling changes in soil occupation  
(figure adapted from [LE 05])

In the example in Figure 2.15, the Agent-Role pattern is found at the level of the association between the plot of land and its occupation of the soil. In this model, an instance of plot of land does not change over time: its surface area remains constant. However, its soil occupation may change. Several types of soil occupation are possible, each of which possesses its own dynamics. A soil occupation of the type cultivation (specialization) can calculate production once its age has reached the number of days before harvest. As for a forest, within this model it has the function of a carbon sink. Using this structure, the operation of a third agent, which uses for cultivation purposes a plot of land previously used as forest, consists of replacing the association of the plot of land to an instance of the class forest with an association to an instance of the class Rapeseed, for example.

Although this architecture is clearly advantageous in terms of its clarity, implementing it within NetLogo can be difficult because only one level of inheritance is possible in this platform, and this level of inheritance is reserved for the *turtle* class. In practice, the modeler is often obliged to specify the attributes and code the operations of all the types of occupation (or roles) for a class, and then to add the conditions which ensure that the entity cannot execute methods which correspond to another type than the one it belongs to at the current time. Also, the modeler needs to ensure that the attributes of a type are correctly reinitialized each time the soil occupation changes.

The above discussion illustrates the difference, however excellent the design, between a model which makes it possible to take all of the paradigms into account and the reality of implementation. This implementation needs to be conducted in an environment or using languages which require a certain vigilance regarding the concepts, or even profound adaptations of these, such as the absence of the concept of objects.

## **2.4. Description and documentation of agent models**

This section addresses the question of describing and documenting a multiagent model. Section 2.4.1 will remind the readers of the roles

that the documentation and standardization of a model fulfill, which are, among many others, making it easier to understand and explain the structure and operation of the model, to spread information about it and communicate about it with others, to compare it, replicate it, to use it for other purposes and to reuse it, and to ensure that it is complete. Section 2.4.2 presents a protocol in detail, which has become a *de facto* standard for describing multiagent models: the ODD protocol [GRI 06, GRI 10]). A brief history of this protocol will be given, outlining the goals that underlied its creation. The various elements which define the ODD protocol (in its revised version) are then discussed: (1) the model's aim, (2) entities, state variables and scales, (3) general operation of the model and process scheduling, (4) underlying design concepts, (5) initialization, (6) input data and (7) submodels. Section 2.4.3 will then illustrate, using various examples, ODD implementation, while highlighting frequently occurring usage or interpretation errors. This section also shows the link which exists between the diagrams created during the model's design phase and the protocol's components.

#### **2.4.1. *Why describe and document?***

The definitions of models and modeling given by Jean-Louis Le Moigne [LE 90] emphasize intelligibility as one of the most important properties of a model:

- A model is an intelligible, artificial, symbolic representation of situations in which we are involved.

- Modeling is the act of intentional development and construction of models, which are likely to render intelligible a phenomenon perceived as complex, and to amplify the reasoning of the actor by projecting a deliberated intervention within the phenomenon; the particular aim of this reasoning is to anticipate the consequence of possible plans of action.

In this way, a model builds an organized set of knowledge which represents a complex system. Questions are being asked about this system; its current operation needs to be understood, and also, possibly, the way that it has developed in the past. Also, its future possible development needs to be known, along with how it is impacted by

various scenarios, and this development needs to be influenced to lead it toward some desirable outcome.

The documentation of a model thus becomes fundamental in the mobilization, understanding and sharing of the knowledge which it brings, and the efficient use of this knowledge. The documentation must make it possible to describe the model in an instructive, explicit and unambiguous manner, which is relevant for as wide a public as possible.

#### 2.4.1.1. *Instructive description of knowledge obtained using the model*

The instructive character of the documentation of a model is in its capacity to address progressively the model's different levels of description, from the most general down to the most detailed. This is because we cannot directly go into the detail or the code of a complex model. As a result, it is essential to have an overview to understand the model's context and aims, to understand what the model is doing before understanding how it is made, and then to focus little-by-little on its global components, and their interactions. Following this, we can progressively move on to the detail of the manipulated data and modeled processes. It is precisely to this end that the ODD protocol was developed. This protocol will be described in the section 2.4.2.

#### 2.4.1.2. *Explaining the structure and operation of the model without ambiguity*

For a model considered stable, none of the aspects should remain implicit or vague to the extent where a knowledge gap exists in understanding the model or rendering it operational. This is a difficulty that frequently occurs when a model is being conceptually refined so that it can be transformed into an operational computerized model. It is this stage which generally makes gaps, ambiguities or incoherencies apparent, which then require a return to the conceptual model and the further redetailing or redefinition of certain aspects of this model. There is a sort of dialectics between the conceptual model and the computerized model which means that several cycles are necessary in order to elicit the knowledge carried by the models progressively. The main difficulty clearly lies in the manner in which the model's documentation takes the organization of the modeled system into

account, the interactions between its components and the properties which emerge from an organizational level in relation to the underlying level(s), all the while managing a record and justification for the decisions taken along the path for attaining a stable version of the model.

In the context of this modeling process, ambiguity takes on a particular character. At the beginning of the model design process, it may be useful to accept some ambiguity in order to avoid undue focus or time spent considering a point which might lead to problems between those parties involved in the modeling. However, it is also important to remove the ambiguity later in the refinement of the design by specifying the conditions which lead to accepting or refusing it. Temporary ambiguity may thus contribute flexibility which is useful for the modeling process, particularly in the case of a process which is being conducted together by several parties.

#### 2.4.1.3. *Taking the multiplicity of model users into account*

The documentation of a model must be relevant to its users at different stages of its development and implementation. Therefore, the model must be made comprehensible to the various stakeholders in the modeling process, which may comprise people with specific knowledge bases (such as scientific, institutional, technical or empirical) and discipline-related interests (thematic or computer-related). A model must be understandable to the various types of users who contributed to its development, as well as to people other than the initial stakeholders, who may wish to use or further develop the model. This involves several levels of documentation, which are not necessarily hierarchical. The use of different forms of documentation (written, graphic, diagram, video, etc.) may also favor the comprehensibility of the model to a varied public. It is necessary to take several points of view into account when building a model of a complex system, and similarly the documentation of a model must make it possible to observe and understand these different points of view.

#### 2.4.1.4. *Including mobilized meta-knowledge when building the model*

The documentation must also explain why certain aspects have been included in the model, and why others have not. Grimm [GRI 99] has remarked that one of the advantages of individual-centered models is that they can integrate empirical knowledge, with all the difficulty of defining the criteria making it possible to decide whether certain empirical knowledge should or should not be fed into the model. It is a requirement of relevant documentation that it should make these criteria clear.

Thus, documenting a model involves more than the mere description of the knowledge it contains; it also requires the description of the meta-knowledge which led to the elicitation of this knowledge (i.e. the model's hypotheses).

#### 2.4.1.5. *Making it possible to replicate the model*

The documentation associated with a model should make it possible to replicate the model. Starting from an analysis of several examples of replication, Bommel [BOM 09] highlighted the inadequacies of descriptions of many models, with particularly weaknesses in specifications relative to the management of time and interactions. He considered this stage of replication to be necessary, and that it should be executed as far as possible by modelers other than the initial designers to enhance the reliability of the simulators and allow a true refutation of models. Literature on the topic has only recently become preoccupied with this question in relation to multiagent models [GRI 06]. A particular reason for this is that their usage has undergone considerable development in several disciplines which handle complex systems. By suggesting a protocol such as ODD, a first stage is available which makes it easier to write and read multiagent models, resulting in a description of these systems which is sufficiently comprehensive to allow them to be replicated. The first revision of this protocol was published in 2010 [GRI 10], and had a basis in the experiences of the many researchers who had begun to use the protocol.

#### 2.4.1.6. *Working alongside the model development cycle*

The question of writing and reading a model has a particular connection with two aspects of its lifecycle: first, how the creation of the model can be facilitated (effective writing), and second, how to facilitate its use and integration into other models (effective reading). These two aspects are not separated in time, but rather are closely linked to each other. Indeed, the writing phase presupposes the involvement of a certain number of people who must be able to share the state of development of the model as it develops, and thus must be able to read the model as effectively as possible. Similarly, once the model is considered to have reached a certain level of completion, being able to read it conditions the writing of new developments of it, or its integration into more comprehensive models.

Complex models can often be divided into a set of submodels which mutually interact, each of which makes up a model in itself, potentially of a nature different from the others. The integration of several models thus requires good description of each of them, which in particular renders the definition of information communication interfaces between them as effective as possible.

#### 2.4.1.7. *Developing relevant forms for visualizing a model*

How can the most relevant possible forms of visualization be developed for making the constitutive processes of the model clear, and for presenting, analyzing and interpreting outputs from the model simulations? Visualization is not only involved in making the conceptual model operational, but it also makes it possible to elicit knowledge for building the conceptual model [BEC 03]. Thus, developing a graphic visualization of a mathematical law associated with a submodel makes it possible to explain and visualize the effects of this law in light of various hypotheses, and, using this as a starting point, to calibrate the model more effectively, or even to alter its structure when required. In this way, comprehension is enhanced by making it possible to find the basis for the results, and to examine the chain of causalities and interactions which led to these, to be able to see the reason for them, and thus to gain a better understanding of the system under study. Graphic interfaces promote communication between the



partners, and can potentially serve to mediate if there is some difficulty in reaching a shared vision. They contribute to the transparency of models, that is to say they help them to be understood quite easily by a wide range of people [WAL 77].

Analysis and understanding of the behavior of models is one of the key factors in their validation. Here, the word “validation” is taken to mean “acceptable for its intended use because it meets specified performance requirements” [RYK 96] and “are well designed and justifiable” [SIN 00].

Visualizing a model entails visualizing the knowledge that it contains, and in particular the knowledge pertaining to its dynamics and organization: static documentation is insufficient for these. In other words, the visualization of a model may be considered to be a dynamic form of documentation. UML and AML, which we have previously discussed, are very good examples of this.

#### **2.4.2. *How can models be described and documented?***

In response to the recurrent difficulty of obtaining complete and homogenous documentation of multiagent models, a protocol has recently been suggested for the description of these models in a more formal manner. This is the ODD protocol suggested by Grimm *et al.* [GRI 06]. ODD offers a predefined documentation structure, which means that modelers can specify an aim for the model, its components and the way in which the properties specific to the MAS are taken into account (e.g. emergence or adaptation). The first version of the ODD protocol was published in 2006 [GRI 06]. After testing within the research community, a revision was offered in 2010 [GRI 10, POL 10, RAI 11]. Today, ODD has become a *de facto* standard for describing and communicating an agent model. It has been adopted by many modelers (e.g. [POL 08, NAI 10, CAI 13]).

##### **2.4.2.1. *ODD protocol components***

ODD describes a multiagent model by making a distinction between three major parts (Table 2.1): the elements that provide an overview of

the model, the design concepts and the model details. We will present each of these elements below in the order recommended by the protocol.

2.4.2.2. *Overview*

The first part of ODD aims to provide a synoptic vision of the modeled system, from the point of view of its structure as well as its dynamics. It is made up of three main elements.

<i>ODD (in English)</i>		<i>ODD (in French)</i>	
Overview	1. Purpose 2. Entities, state variables and scale 3. Process overview and scheduling	Vue d'ensemble	1. Objectif 2. Entités, variables d'état, échelle 3. Processus et ordonnancement
Design concepts	4. Design concepts <ul style="list-style-type: none"><li>– Basic principles</li><li>– Emergence</li><li>– Adaptation</li><li>– Objectives</li><li>– Learning</li><li>– Prediction</li><li>– Sensing</li><li>– Interaction</li><li>– Stochasticity</li><li>– Collectives</li><li>– Observation</li></ul>	Conception	4. Eléments de conception <ul style="list-style-type: none"><li>– Principes</li><li>– Émergence</li><li>– Adaptation</li><li>– Objectifs</li><li>– Apprentissage</li><li>– Prédiction</li><li>– Perception</li><li>– Interaction</li><li>– Stochasticité</li><li>– Coopération / Agrégation</li><li>– Observation</li></ul>
Details	5. Initialization 6. Input data Submodels	Détails	5. Initialisation 6. Données d'entrée 7. Sous-modèles

**Table 2.1.** *ODD protocol components divided into three categories (as per Grimm et al. [GRI 06, GRI 10])*

2.4.2.2.1. *Purpose*

The purpose of the model is indicated in a concise but precise manner. What is it for? What is the modeling question that is being addressed? The protocol component defines the “What?”.

#### 2.4.2.2.2. Entities, state variables and scale

Once the purpose of the model has been declared, its various components are specified. Which entities are represented? What are the attributes that characterize these entities? What spatial and temporal scales are used? In other words, “of what” is the model made?

The entities involve all the categories of the model (agents and groups of agents, spatial entities and global environment). These are described in terms of properties, known as state variables in ODD, which have values that may or may not evolve during the simulation (e.g. an agent’s name, the behavior strategy of a group of agents, the modality of soil occupation of a plot or the global rate of harvesting of a resource). The initial values are provided at the model’s initialization. These state variables (quantitative, nominal qualitative or ordinal) should not include derived or aggregate variables, whose values result from a combination or a calculation conducted by using other variables (e.g. total quantity of food consumed during the simulation, average animal density per hectare or distance to the nearest neighbor).

As for the choice of spatial and temporal scales, this involves, on the one hand, specifying the spatial resolution of the patches or the area actually covered by the patch and, on the other hand, the temporal resolution or the real duration represented by a time-step in the model. The temporal horizon is also specified (duration or planned length of the simulation).

#### 2.4.2.2.3. Process overview and scheduling

Following the structure, the model dynamic is described. How do the entities behave at each time-step? How does the environment change? What order is the simulation organized into? The set of processes and the way they are scheduled are mentioned.

Processes are designated by verbs (e.g. moving, fleeing and updating the quantity of grass consumed) and they refer to the submodels listed in the third part of the protocol. The operation properly speaking of the processes is thus not provided at this level. The list of actions carried out and the order in which they occur are all that are provided at this stage.

The actions are those carried out by the models entities (the agents or the patches) and also those carried out by higher level entities such as the model itself or the observer (for updating global indicators and the associated graphics).

The ODD protocol does not impose any particular formalism for the presentation of this varied information. However, it is advisable to use a UML class diagram to describe the model's structure (see section 2.3.1.1). The scheduling of the process may also be described using a UML activity diagram (see section 2.3.1.2).

#### 2.4.2.3. *Design concepts*

Elements of documentation which are important for interpreting results are information on the emerging properties of the model, the hypotheses put forward and the capacities of certain agents (e.g. adaptation or learning). These elements are often not particularly formalized. In this part, ODD suggests a list of 11 items which provide information, not on the operation of the model, but on the way in which the concepts specific to the modeling have been taken into account. All of these concepts can be made explicit through answering the various questions suggested by the protocol, which are in the form of a checklist [GRI 10].

The ODD protocol makes it possible not to provide information about all of the concepts suggested in this part because some of these concepts may not be appropriate to the model that is being developed. Also, it is possible to add certain concepts unique to the user. However, in this case, it is advisable to explicitly specify that these are not elements that are provided by the standard.

#### 2.4.2.4. *Details*

After a sketch of the model, with its main dynamics, has been created, this third part addresses the technical details which should make it possible to reimplement that model as a whole. ODD offers information provision at this level for three elements: initialization, input data and submodels.

<b>Principles</b>	What are the concepts, hypotheses or theories underlying the model design? At what level are they integrated into the model?
<b>Emergence</b>	What are the emerging concepts not foreseen by the model, which result in interactions between or adaptations among agents? What are the expected emergent phenomena which result from the rules introduced into the model?
<b>Adaptation</b>	Do the agents always retain the same behavior? Are they able to adapt during the simulation? Do they have the choice to behave according to several alternatives? What are the decision-making rules which regulate this choice? What are the conditions for a potential change in behavior?
<b>Goals</b>	Do the agents seek explicitly or implicitly to reach a goal in relation to their adaptive behavior? What is this goal? What is its indicator/criteria? What is the usefulness/fitness function?
<b>Learning</b>	Does the experience acquired by the agents during the simulation make their decisions change? Are they able to learn? How are these learning mechanisms implemented?
<b>Prediction</b>	Can the agents evaluate the consequences of a decision that they might make? How do they predict the effect of their decision? Are they able to do this?
<b>Perception</b>	To what information do the agents have access? What are the state variables that they perceive or receive from other agents (internal variables or variables relative to the environment)?
<b>Interaction</b>	What direct or indirect interactions are integrated into the model? Upon what mechanisms present in the reality of these interactions are they based? Are they local or global interactions? Does the model allow the agents to communicate? In what form?
<b>Stochasticity</b>	Which model processes or variables introduce a random element? Why is this random element represented?
<b>Co-operation / aggregation</b>	Is there an organizational level in the model which is made up of groups of agents? Are these group products of an emergence phenomenon or explicitly defined because they share common properties (notion of breeds in NetLogo)?
<b>Observation</b>	What are the indicators observed during the simulation for the understanding and the analysis of the model's behavior? What are the outputs (e.g. data or graphs)?

**Figure 2.16.** *The key questions ODD proposes in order to specify the agent model's design elements (taken from [GRI 06, GRI 10])*

#### 2.4.2.4.1. Initialization

The documentation should specify all the simulation's initial conditions. How many agents are there? How are they distributed in space? What are the initial values of the variables and the parameters? What is the state of the environment? Are the initialization conditions constant and based on reference data, or are they stochastically fixed? The idea here is to provide all the information required for the reproduction of the results of a simulation.

#### 2.4.2.4.2. Input data

The dynamics represented in the model may be based on auxiliary data (for example, an imported predefined spatial environment), they may be modulated by forcing factors (for example, relief, surface temperature and quantity of precipitation) or they may integrate an existing model (for example, the growth curve of a plant species). If this is the case, this is the level where it should be mentioned. If not,

the protocol recommends that it should be explicitly indicated that the model does not call for particular input data. It should be noted that this documentation element does not concern the value of parameters or state variables, which may also come from external files.

#### 2.4.2.4.3. Submodels

The processes that make up the model and their order of execution were specified in the first part of the ODD. In this part, the operation of each of them should be precisely indicated, as well as the reasons why certain hypotheses were adopted, and how these submodels were calibrated, along with any usage limitation they may have. The submodels may be composed of equations, algorithms or specific rules, all the parameters of which must be explained and justified. If the submodels are based on theories or methods that have already been published, these publications should be referenced.

### 2.4.3. *ODD documentation of the recurrent example*

In order to illustrate the use of the ODD protocol, the recurrent model will be presented below, documented according to this standard.

#### 2.4.3.1. *Overview*

##### 2.4.3.1.1. Purpose

The aim of the model in the recurrent example is to explore the propagation conditions for malaria in the subregion of Maroua (Cameroon) in light of the pendular mobility patterns of farmers, who make a daily journey between their homes (in town) and their plots of land (outside town). The goal is to understand and measure the impact of these journeys on the spread of the illness.

##### 2.4.3.1.2. Entities, state variables and scale

The model is made up of two agent categories: humans (farmers) and mosquitoes. These two types of entities are characterized by a state of infection (`is-infected?`) and an attribute which indicates whether the agent is a source of the epidemic or not (`is-infection-exterior?`). Each agent is also located in space (attributes `xcor` and `ycor`). Human

agents also know the place where they work and where they live (attributes `home` and `work`) as well as the time when work begins and when it ends (attributes `beginning-work` and `end-work`).

The environment in which the agents are situated is made up of a set of places (spatial units) which are characterized by a soil occupation (attribute `nature` of the class `Place`) linked to a color (attribute `pcolor`). For each place, it is specified whether the place is a dwelling place or not (attribute `place-dwelling?`). The spatial units correspond to 50 m x 50 m cells which form a rectangular grid, representing the territory (dimension: 523 x 424 cells). These dimensions are imposed by the map taken as input data (see the Initialization of the ODD description part).

As for temporal resolution, the simulation time-step is set at 10 min (which makes 240 simulation steps per day). The model is built to carry out simulations of up to 30 days.

Several variables are also defined as model parameters: the number of mosquitoes present in the environment (`number-mosquitoes`), the number of farmers (`number-humans`), the distance from which a mosquito can reach farmers to bite them (`distance-contamination`) and the home-work distance for each farmer (`distance-home-work`). The model makes provision for an activation or non-activation of mosquito reproduction (`reproduction-mosquitoes?`) as well as for possible contagion between farmers and mosquitoes during the journey between home and work (`contagion-transport?`).

A simplified presentation of the model structure is given in the UML class diagram (Figure 2.2). The set of state variables is reproduced in Table 2.2.

#### 2.4.3.1.3. Process and scheduling

At each time-step, the processes are conducted in the following order: (1) the mosquito agents act by moving randomly through space, biting a human present in the radius of contamination (procedure `go-mosquitoes`) and (2) depending on the time of day, the humans go at work (procedure `goto-work`), stay at work and go back at home at the end of the day (procedure `goto-home`). These farmer actions

are included in the procedure `go-humans`. Within each set of agents, the simulation platform chooses the order in which the agents carry out their tasks. At the end of each time-step, the simulator updates the different output indicators: the number of mosquitoes and farmers infected according to the time.

The dynamics of the patches is limited to the creation of new mosquitoes (`go-patches`).

<i>Entity</i>	<i>Variable name</i>	<i>Possible values</i>
Human and mosquito	<code>xcor</code>	[0,523]
	<code>ycor</code>	[0,424]
	<code>is-infected?</code>	{true,false}
	<code>is-infection-exterior?</code>	{true,false}
Human	<code>home</code>	one patch
	<code>work</code>	one patch
	<code>start-work</code>	[0,240]
	<code>end-work</code>	[start-work,240]
Patch	<code>pcolor</code>	[0,240]
	<code>place-dwelling?</code>	{true,false}
Global	<code>step-time</code>	[0,2400]
Parameters	<code>number-mosquitoes</code>	[0,1000]
	<code>number-humans</code>	[0,2000]
	<code>distance-contamination</code>	[0,10]
	<code>distance-work-home</code>	[1,500]
Patch	<code>contagion-transport?</code>	{true, false}
	<code>reproduction-mosquitoes?</code>	{true,false}

**Table 2.2.** *Summary of the different state variables and parameters*

### 2.4.3.2. Design concepts

#### 2.4.3.2.1. Principle

It is hypothesized that the human agents perform a pendular movement: during the day, they begin traveling to their workplace, remain immobile at the workplace in order to work and then return to



their homes. In addition, various studies have shown that mosquitoes are present throughout the territory and that their range of movement remains very local. This is why the hypothesis that they move randomly within a limited radius is realistic.

#### 2.4.3.2.2. Emergence

A spatial propagation phenomenon for the disease is observed, which follows the movement of the human agents.

#### 2.4.3.2.3. Perception

The mosquitoes have the capacity to perceive nearby farmers (that is those who are present within a radius that is less than the parameter *distance-contamination*). As for the farmers, they are aware of time, and this helps them decide whether it is time to go to work, to stay in work or to go home. However, they cannot perceive the other farmers or the mosquitoes.

The cells know whether the mosquitoes are located on them (in order to decide they are producing a new mosquito agent or not).

#### 2.4.3.2.4. Interaction

The only interactions present in the model are the interactions between a human agent and a mosquito agent. When a mosquito perceives a farmer within its contamination zone, it will bite that farmer. If one of the two agents is a carrier of malaria, the other will be infected. If both agents are in the same state of contamination, the bite will have no effect.

#### 2.4.3.2.5. Stochasticity

The model contains a share of stochasticity, both in the initialization and in the dynamics. In particular, the position of the mosquitoes in space (with respect to the humans) is random for one of the cells (with respect to dwelling places). The position of the human workplace is also random among the cells that are not dwelling zones. The first infected mosquito is also chosen at random. During the simulation, two dynamics include an aspect of randomness. During the creation of a new mosquito, it is placed onto one of the neighboring cells (chosen at

random) of the wetland cell that “produces” the mosquito. In addition, the movement of the mosquito is also partially random: it moves at a constant distance but in a direction that is chosen at random.

#### 2.4.3.2.6. Observation

The main display of the simulator is a map showing the land occupation types (black cells for dwelling places, blue for wetlands and black for farming land), along with the mosquitoes and humans. This display is updated at each time-step, allowing the observer to see the dynamics of the movements. In addition, the graph of interactions between humans and mosquitoes (bites) is also displayed, and is updated at each time-step.

During the simulation, the development of the number of mosquitoes and humans can be seen through a curve which shows these values over time.

The simulator can also calculate and display (when this is required by the user) the average number of human–mosquito interactions (i.e. the number of mosquito bites) which have occurred before a human is infected.

#### 2.4.3.3. *Details*

##### 2.4.3.3.1. Initialization

The first stage of simulation initialization is the loading of the map, which sets the color (and thus the usage type for the soil) for the different cells. Following this, the mosquitoes are randomly created throughout the space with a healthy infectious status (`is-infected?` and `is-infection-exterior?` are initialized at false). The human agents are then randomly created at dwelling zones, with a workplace chosen randomly in the non-dwelling zones and a healthy infectious status. Finally, an epidemic is created: one of the mosquitoes is chosen at random and becomes infected. It is also noted as being the source of the epidemic (`is-infection-exterior?` is set as true).

##### 2.4.3.3.2. Input data

As an input, the models take a map of soil use in the Maroua subregion, in the form of a raster image. Only three types of soil use

are shown on this map: dwelling place (in black), wetland (in blue) and farmed land (in white).

#### 2.4.3.3.3. Submodels

We can make a distinction between three submodels within the model, which correspond to the three main dynamics of the model:

*pendular movement*: the behavior of the human agents is limited to their pendular movement – each day, they go to work in the morning, work (and thus remain immobile) during the day and then return home in the evening. The choice between these categories is made as a function of the simulation step during the day (i.e. as a function of the simulation step). We have refined this model to allow for infection during the journey (parameter `contagion-transport?`): when `contagion-transport?` is false, in order to avoid any contact between humans and mosquitoes during the journey, we consider that the human displacement is instantaneous (whereas, it takes several simulation steps when `contagion-transport?` is true);

*demography of the mosquitoes*: we have introduced a very simple reproduction dynamics for the mosquitoes, based on the fact that they lay eggs in water and spend the start of their life (as an egg, larva and nymph) in the water: each day (and thus every 240 simulation steps), the wetland cells (which are blue) will create a mosquito agent if another mosquito agent is present in the cell;

*infection*: at each simulation step, all the human agents who are within the contamination radius of a mosquito are bitten. For each mosquito–human interaction, if the infectious state of one of the two agents is infected and the other is healthy, then the healthy agent becomes infected. If both are either healthy or infected, their infectious state remains unchanged.

## 2.5. Discussion on documentation

Nowadays, modelers have many languages available to them for formalizing and documenting their MAS.

UML is well suited to the description of the structure and the behavior of the model. It remains more widely used and offers a semantics which is rich enough to describe many MAS. In the case of models which include more complex organizations or actions, the use of AML, enriching the UML notation, can be particularly useful. In any case, an MAS should be described at the very least by a class diagram, which describes its structure, and by activity or state-transition diagrams, which model the behavior of the entities.

As for the documentation, the ODD protocol is becoming increasingly widely used by modelers, especially for presenting multiagent models in research literature. It is gradually becoming a recognized standard, on the same level as UML. It may also be noted that the ODD descriptions include increasingly UML class diagrams for explaining the structure of models being presented graphically and synthetically, which demonstrates the complementarity of the tools. ODD offers predefined documentation structure (the packaging), whereas UML allows the production of a part of the content in the form of diagrams.

The designer's attempt to formalize and document the model is essential in the lifecycle of the MAS. It makes it possible to use the model in a more reliable and enlightened way, so the users can gain a better understanding of the questions it attempts to contribute to answering and can see precisely what structure and operation of the representations are use to achieve this. However, much more remains to be done in this field not in terms of the design, but in terms of the use of models, first, in order to increase their reliability and, second, to facilitate informed use of them:

– *make models more reliable to use*: this will involve the transition from a documentation of a model as the result of a modeling process to a documentation of the modeling process itself. It will include the justification of choices made and test methodologies used to ensure that insofar as possible the computer implementation of the model is faithful to the conceptual model that was developed. This aspect also includes a quality documentation of the usage conditions of a model so that it is used advisedly to address the question that has been targeted throughout

its development, and so that the risks of abusive use to resolve problems to which it is not suited are avoided;

– *make models easier to use*: it is also becoming important to facilitate the uptake and analysis of a model's results, for example, by the definition of the different usage scenarios and a detailed analysis of the results obtained.

Recent studies have shown that this question is beginning to be addressed, for example, with the development of the tool *Transparent and Comprehensive Ecological modeling documentation* (TRACE) [GRI 14], which offers standardization of the documentation associated with the models' objectives, with their design and with their implementation.

It is now more important than ever to provide conditions which favor the establishment of connections between the designers and users of models, in a process of mutually working alongside each other.