From the problem to the implementation: a method associated to a model to design Multi-Agent Systems

Bruno Mermet*
Laboratoire Informatique du Havre (EA3219)
25 rue Philippe Lebon
BP 540
76058 Le Havre, France
Bruno.Mermet@univ-lehavre.fr

Gaele Simon†
Laboratoire Informatique du Havre (EA3219)
25 rue Philippe Lebon
BP 540
76058 Le Havre, France
Gaele.Simon@univ-lehavre.fr

Dominique Fournier‡
Laboratoire Informatique du Havre (EA3219)
25 rue Philippe Lebon
BP 540
76058 Le Havre, France
Dominique.Fournier@univ-lehavre.fr

ABSTRACT
This paper deals with a method and a model called SPACE allowing to design multiagent systems. Their main interest is to introduce tools to design and to validate the produced system at the same time. First, the main steps of the proposed method are described. Then, the different components of the SPACE model are defined. Finally, a case study (the graph colouring problem) shows how the method and the model can be applied.

Keywords
multi-agent system, method, agent model, MAS validation, graph colouring

1. INTRODUCTION
For the last decade, in order to answer to the growing interest for multi-agent systems (MAS) development, several methods, models and tools have been proposed [17, 23, 14]. Among them, there are some methods [27, 12], agents models [21, 8], MAS models [16] and MAS development environments like Madkit, Jade, Zeus... [15, 22, 20]. Our research focuses on the design of a MAS development method coupled with a model allowing, at the same time, validation, verification or even system proof. Generally, already proposed methods do not provide enough necessary elements to do this task. Moreover, we want the method to be a real guide for the developer to help him to analyse and to break up the global goal of the MAS so as to obtain the structure of the MAS, the different agents, their goal and their behaviour. Most of existing methods propose only a set of models which can be used to express different aspects of the MAS but do not really propose guidelines to apply these models to the problem which must be solved by the system. Finally, we wish the method to be coupled with models in order to be able to directly implement specifications obtained by the method. Unfortunately, models proposed by the existing methods can not often directly be used to implement the system in a MAS development platform.

In [19, 24], we have described a first development of our method dedicated to optimisation problems. This method is summarised in the paper. Starting from an informal problem specification, we show how to obtain the specification of a set of agents allowing to solve this problem. For this process, we introduce specification tools such as variants which will be used, in the future, to help to verify properties of the developed system. Moreover, this method allows to associate a goal to each agent whose behaviour is described by an automaton.

As for other methods, using this method allows to produce an operationalisable specification of each agent of a system. In existing methods, this specification can be expressed in different formalisms like A-UML, Gaïa... In every cases, it must be transformed into an operational specification which can then be executed. We propose a model which can be used to make this task easier. This model, called SPACE (Structure, Perception, Action, Comportment and Environment) can be seen as an interface between agents model, like BDI, Aalaadin, Vowel [21, 8, 3]... used to characterise agents behaviour, and their corresponding implementation in a MAS development environment. For example, the BDI model can be expressed using SPACE model [13]. Once the SPACE specification of the agents has been obtained, their implementation becomes an automatic task. Indeed, with the SPACE model, we provide a direct translation into several MAS development environments like MadKit. So, the SPACE model has been designed to be used both to specify agents produced by our method and to implement agents

*Dr. Mermet is senior lecturer at the university of Le Havre. He is a formal methods and MAS specialist.

†Dr. Simon is senior lecturer at the university of Le Havre. She is a Knowledge management and MAS specialist.

‡Dr. Fournier is senior lecturer at the university of Le Havre. He is a machine learning and knowledge discovery specialist.

produced by other methods and expressed in higher-level models.

Using SPACE model allows the developer to avoid the coding step into the MAS development environment language he wants to use. Moreover, a SPACE specification of an agent gives the most declarative operational view as possible of this agent. This property gives the basis to potentially future validations of the agent implementation by reasoning on its specification in SPACE. Last but not least, this model must allow to control the consistency of the agent specification.

Indeed, we propose a MAS validation process in four steps :

1. Inner consistency verification of each agent model :
   the SPACE model described in this paper, must provide tools to do this verification. For example, it must be checked that all variables used by the agent perceptions are defined in the agent model.

2. Inner consistency verification of the MAS model :
   this verification must not only take into account all the agents models but also relations between agents and the number of instances of each agent model. Suppose for example that a given agent, instance of A model, can make its job only if n agents, instances of a second B model, are created. Our model must allow, in this case, to verify that the MAS can always create these n instances of B model.

3. Agents behaviour validation :
   the problem, here, is to be able to answer to the following question : "does the agent really act as the developer wanted to ?". In other words, does the effective behaviour of the agent fit to the specified behaviour ?

4. System behaviour validation :
   for this validation, the question to which an answer must be given is the same as the previous one but at a different scale : the entire MAS. As a consequence, this validation allows to verify that the global behaviour of the system fits to the system behaviour the developer wanted to obtain.

The inner consistency verification is an aspect that as already been taken into account in the model and that is presented in the paper. Our research is currently focused on agent behaviour validation. It is one of the reasons of the creation of the SPACE model. Indeed, we need a model that has enough formal materials for validation. We are now working on the validation process itself, that is to say, specifying how to combine specific properties we have put in SPACE in order to perform the validation.

The first part of the paper is dedicated to the description of the method we propose. Then the SPACE model is presented. After these two first parts, we show how the method and the model have been used to build a multi-agent system for the graph colouring problem whose main principles and constraints are briefly presented.

2. METHOD

The goal of our research is to have a method and also tools to help the analysis and design of MAS.

As presented in the sequel, the method is based on a top-down approach which warrants the progress of our system towards a *satisfying* solution.

2.1 Usage conditions

The method defined here must be used to solve global problems which can be specified by a set of local constraints (LC). A more restrictive usage condition is that this method is dedicated to problems for which a trivial (but probably bad) solution exists.

Of course, for non NP-hard and non distributed problems for which a sequential algorithm is known, using agents (and so our method) is rarely a good solution because communications and synchronisations introduced by MAS make the program less efficient [26].

An example of a target problem for our method is the graph colouring problem which consists in colouring a graph with a minimal number of colours in such a way that two connected nodes do not have the same colour. This application is presented in section 4.

2.2 The method

2.2.1 *Global variant*

The first thing to do is to define a variant characterising the problem that the system must solve. A variant is a notion often used to prove termination of algorithms. A variant is a variable defined on a totally ordered structure that must decrease at each iteration and that has a lower bound. These two properties imply the termination of the iterations.

2.2.2 *Local decomposition*

The second step is perhaps the hardest one: the global problem has to be expressed in terms of local sub-problems. This consists in dividing the solution of the problem into several parts with respect to LC. These parts are not necessarily disjunctive. Each part is associated to a local sub-problem. The resolution of each of these sub-problems must help to solve the global problem. The ideal case is a sub-problem whose resolution is a necessary condition for solving the global problem. However, this is not always the case. An other possibility is a sub-problem whose resolution makes the global variant decrease.

2.2.3 *Agentification*

Once the global problem has been decomposed, we still do not have agents. Of course, a first idea could be to assign each local problem to an agent, but this is not always possible. Indeed, to agentify a problem :

- each sub-problem must be assigned to an agent;

- each property of the problem (piece of data) must be assigned to an agent.

However, each agent perceives only a local part of the environment. Moreover, an agent being autonomous, no other agent can modify directly its properties. These two constraints are called *the locality principle*. So, if the resolution of two sub-problems needs to modify the same property, assigning two problems to two different agents is impossible. A first solution could be to assign properties to the environment instead of agents. This is an easy solution, but this makes the environment a central resource for the MAS, limiting the benefit of the distribution.

A better solution is to change the structure of the local problems so that the modification of a property occurs only in the resolution of one sub-problem. So, each property modification is controlled by one and only one agent. Other agents that need to get the value of this property must have the agent owning the property in their accointance set and can know its value by message passing. Subproblems resulting from this restructuring are called Property Oriented Sub-Problems (POSP) in the sequel.

This step is necessary (it provides the agents and the accointance relations of the MAS) and not so difficult to realize as it is shown in this article for the graph-colouring problem.

2.2.4 Agents behaviour

2.2.4.1 General behaviour

Each POSP is assigned to an agent. So, the general behaviour of each agent is very simple:

- if its problem is solved, it does nothing (it could also help other agents). The agent is *satisfied*;
- otherwise, it tries to find a solution to its problem.

2.2.4.2 Solving a problem

For the global problem, we introduced a variant. We have to do the same for each POSP in such a way that each time a local variant decrease, the global variant does not increase. These variants allow to control the progress of each POSP solving.

Each POSP can be divided into sub-goals whose resolution makes the local variant decrease. Then, an unsatisfied agent chooses a sub-goal and tries to solve it.

When a sub-goal has to be solved, there are two cases:

- either it can be solved by the agent: the agent can then choose a new goal;
- or the agent cannot solve it.

There are two reasons making a subgoal unable to be solved:

- either there is a blocking situation: an other agent prevents the active agent to apply one of its strategies;
- or the agent doesn't know what to do to solve the subgoal in the given situation.

In the second case, the agent chooses an other goal or waits for a modification of the situation. In the first case, the agent attacks an obstructing agent. This behaviour follows the eco-agent's one [7]. The attack mechanism is simulated by sending an aggression message. An agent under attack has to flee so that the blocking situation disappears, but preserving the local constraints LC. Note that the fleeing behaviour can increase the local variant. If the agent cannot flee, it ignores the attack.

In order to help us specifying agents behaviours, we used the formalism of transducers [1]. This formalism can also be used to specify behaviours of other kinds of agents [18]. Thus, we define the general behaviour of an eco-agent by a transducer as detailed in [24].

The method described in this part helps MAS developers to determine and specify the set of agents (and also their goals and behaviour) which are necessary to solve the initial problem. The next part describes a model allowing to design these agents from an implementation point of view.

3. THE SPACE MODEL

In this part, we present the different elements of the SPACE agent model. As explained before, this model is designed in order to be a good compromise between a purely executable agent specification about which it is not possible to reason and a more formal specification not really implementable. We try to obtain an almost runnable agent specification while keeping the most declarative description as possible. As a consequence, the resulting specification can be considered as an operational one: we provide not only models but an operational semantics describing the effective operation of the agent from these models.

In SPACE, an agent is represented by three complementary models and a structure: the perceptions model, the actions model, the behaviour model and the inner structure.

These different elements are described in the following paragraphs. They are of course not independent from each other. That's why they must satisfy a set of consistency rules which are also described in further section (3.6).

3.1 Perceptions model

This model describes the set of perceptions of the agent, that is to say, whatever is likely to modify the actions the agent may do. There are four different perceptions which can be divided into two kinds:

- Message perceptions: they occur when a message is received from another agent. There are two kinds of messages associated to such perceptions:
  - information messages: an Information Message Perception (IMP) is associated to each kind of information messages,
  - request messages: a Request Message Perception (RMP) is associated to each kind of request message,

- Variables Perceptions: they represent the agent view of some variables. They are also divided into two kinds of perceptions:
  - Environment Variables Perceptions (EVP),
  - Inner Variables Perceptions (IVP).

3.1.1 Variables Perceptions (EVP/IVP)

These perceptions are connected to inner and environment variables. The environment in which the agent is living is considered as a set of variables. The environment evolution implies changes on the values of the variables. A priori, an agent has only a partial and biased view of the environment. The prey-predator model is a good example of the difference between the environment and its perception by an agent. A predator only knows a part of the grid on which it moves. This vision is also biased because it depends on the agent interpretation. By this way, it is easy to take into account the point of view of the agent in its behaviour. Thus, some elements of the environment may be considered differently according to the goal of the agent. For example, let us consider weather variables like wind, hygrometry and temperature. Using these variables, an agent can have a perception about the current weather. In an airport, assuming the wind blows slowly, an agent in charge of planes take off can discern propitious conditions. Even though, simultaneously, another agent dedicated to control of a wind farm would estimate the weather conditions are bad.

An Environment Variable Perception associates a particular value to a subset of values of significant environment variables for the agent. It allows to design the agent point of view on the environment. Considering again the three weather variables, the perception "weather" could have four values: nice, very nice, bad and very bad. Thus, when the wind speed is lower than 15 MPH, the hygrometry fewer than 100% and the temperature between 20°c and 25°c, the weather perception shall be nice.

Inner variables perceptions rely on the same principles, but they are not based on the same kind of variables. As it will be presented later, an action performed by an agent rely on perceptions. As a consequence, we decided to introduce inner variables perceptions in order to allow the agent to perform actions according to its own life cycle (its state, its goals, ...), and not only in reaction to environment variations. Inner variables perceptions are very important ones because they allow to define proactive behaviours. To define such perceptions, it is assumed that an agent maintains some variables during its life. Generally, these variables are connected to the problems and goals it has to achieve. They are designed to manage some aspects of the agent behaviour. If we consider again the agent in charge of the wind farm, it shall have at least two variables indicating the number of wind turbines and the number of running ones. Each inner variables perception allows the definition of significant subsets of variables values which could modify its behaviour. The agent in charge of the wind power plant shall have a perception "free turbines" with the value corresponding to the difference between the total number of turbines and the number of running ones.

Formally, the table 1 describes the 5-tuple (N, C, V, T, F) defining variables perceptions.

Name	Type	Role
N	String	Name of the perception
C	{EVP, IVP}	Kind of the perception
V	{variables}	Set of significant variables
T	{values}	Set of potential values
F	Function	Function from V to T describing the value of the perception according to the value of the concerned variables.

Table 1: EVP and IVP

3.1.2 Information Messages Perceptions (IMP)

These perceptions are based on the information messages. These messages contain information sent to the agent (the receiver). An information message is considered as a set of variables. Receiving these variables may modify a subset of receiver's inner variables. Yet, every values are not necessarily useful to the receiver. An IMP allows to define which elements of the message are relevant for the receiver. The information message also provides the way to read and use the information it contains.

These perceptions are defined by a 6-tuple (N, C, S, V, T, F) given in the table 2.

Name	Type	Role
N	String	Name of the perception
C	IMP	Kind of the perception
S	Class	Message structure (name and type of each variable)
V	{inner variables}	List of inner variables the perception may modify
T	{true, false}	true if the message of class S is in the mailbox; false otherwise
F	Function	Casting function from S to V describing how to modify inner variables according to the value of the message.

Table 2: IMP

3.1.3 Request Messages Perceptions (RMP)

These perceptions rely on request messages. These messages allow the sender to ask for information/action to the receiver. As for information messages, these information may be represented by values of variables. So, if the receiver decides to take the request into account, the request message must be saved in the inner structure of the agent in order to be processed later.

RMP are defined by a 6-tuple (N, C, S, F, T, A) detailed in the table 3.

This is worth noting that the sender may be (and will often be) a field of the message. This allows the receiver to deter-

Name	Type	Role
N	String	Name of the perception
C	RMP	Kind of the perception
S	Class	Structure of the request message
F	Function	Function defined on S and inner variables with values in {true, false} specifying if the request must be taken into account or must be ignored
T	{true, false}	true if the message of class S is in the mailbox; false otherwise
A	Action	Action responding to the request (by a message sending or not). The associated event (§ 3.2) must determine when the agent is able to answer.

Table 3: RMP

mine, for instance, which agent it must send the answer to, or what sort of relationships it has with the sender.

3.2 The actions model

The action model allows to describe the entire set of actions the agent is able to do. It also allows to describe each action, to describe conditions in which it can be performed (precondition), and also to describe properties (on inner or environment variables) always having to be checked after its execution (postcondition). Actions may consist in reactive behaviours (for instance, in response to a Request Message) or in proactive ones (i.e. depending on inner variables related to the current goal of the agent). We can notice that sending messages is considered as specific actions. The associated precondition describes, in this case, the time when the agent can send the message.

An action is then defined by a 3-tuple (Pre,M,Post) described in table 4.

Name	Type	Role
PRE	Event	Precondition. Precondition are specified by events (see below)
M	Method	Describes what the action does.
POST	{properties}	Postcondition expressed as a set of properties. A property is a constraint over one or more inner environment variables.

Table 4: Actions

Definition: An *event* is a (named) function defined on values of one or more perceptions or events, with result in B = {True,False}. This notion allows to specify a significant special context for the agent behaviour, i.e. in which it can implement specific actions.

For example, if an agent wants to play tennis, several conditions have to be checked. To express them, an event "tennis condition" will be defined. This event takes into account values of perceptions as "weather" (see § 3.1.1) and "free partner". Thus, this event will be "true" if the "weather" perception value is "nice" or "very nice", and if the "free partner" perception value is "true".

In a more proactive context like the pray-predator problem, a predator shall hunt if and only if it is hungry. As a consequence the hunting action may depend on an event relying on an inner variable perception using a variable evaluating the feeling of hunger of the agent.

3.3 The behaviour Model

A part of the agent behaviour is formalised by specific automata which are transducers.

A transducer is defined by a 6-tuple t = (Σ, γ, Q, I, F, δ) such that:

- Σ is a finite input alphabet,
- γ is a finite output alphabet,
- Q is a finite set of states,
- I $\subseteq$ Q is the set of initial states,
- F $\subseteq$ Q is the set of final states,
- $\delta \in (Q \times \Sigma \times Q \rightarrow \gamma)$ is the application associated to transitions.

To adapt this tool to specify the agent behaviour, we made specific choices concerning the part assigned to each state and the labels associated to the transitions [19, 24]. Within the framework of our model, the agent automaton is in fact the representation of a decomposition of its goals. More precisely, a goal (or sub-goal) is associated to each state. A transition between two states p and q is labelled by a couple (E,A) where E is an event which can occur when the agent decides to leave p. If E occurs, the agent executes the action A and tries to reach the goal q. We give below the state and transition patterns with more details.

A state p is defined by a 7-tuple (N,L,M,Ve,Vs,V, π) such that:

- N is a string labelling the state,
- L={(E,A)} is an ordered list of couples (event,action). Notice that an action A can be performed only if the associated event E is true.
- M is a method, possibly neutral, which sorts actions of L if a specific order is needed,
- Ve is the set of input variables, i.e. variables the agent can read when it tries to reach the goal p,
- Vs is the set of output variables, i.e. the ones it can modify during its stay in state p,
- V is the variant of the goal p,
- π is the choice policy of transitions (ordered list) starting from state p when several events are true at the same time.

Transitions are defined by a 4-tuple (Ed, Ea, E, A). Such a transition shall be fired when the current state is Ed and when event E is true. The action A will then be run and the current state becomes Ea.

3.4 Inner structure of the agent

The inner structure of the agent contains a set of variables which represent, in real time, the evaluation the agent can do of its own situation. In fact, it corresponds to its knowledge of itself. In this structure, there is a list of known agents and a memory of received requests. There are also inner variables, especially those which interact with IVP and IMP.

The list of known agents shall be linked with a qualitative judgement about the kind of relationship (friends, enemies,...). This could be taken into account, for example, in the management of RMP decision process. Last but not least, the inner structure includes the behaviour automaton, and a reference to the current state in which the agent is.

3.5 Operational semantics

The basic behaviour of the agent is described by the automaton. The agent's behaviour in a given state depends whether a method is associated to it or not. If no method is associated to the state then the list L of pairs (event E, action A) is processed sequentially else the list L is processed in the order given by this method. For each pair (E,A), the agent first evaluates the event E. This event relies on perceptions Pi that are also evaluated at this moment. The evaluation of messages perceptions implies the checking of the agent mailbox at this moment too (messages corresponding to the perceptions which are found at this moment are removed from the mailbox). Messages that do not concern the perceptions Pi are preserved but not processed. If this event E is true, the action A triggers (and is not executed if E is evaluated to false). Notice that many actions may depend on the same event. For instance, the list L could look like { (e1, a1), (e2, a2), (e1, a3) }. So, the event e1 will be evaluated twice. It may be true the first time, and be false the second one. We can make another remark: if the action a3 must be performed only if a1 was performed, then the event associated to a3 must not be e1 but a new event e3 associated to an IVP relying on an inner variable set by the action a1 to tell it was executed.

Once all the pairs of the list L associated to the current state have been processed, the agent analyses if the goal associated to the current state is reached (has the variant reached its lower boundary ?):

- if the goal is completed, one of the transitions whose original state is the current one must be fireable (a fireable transition is a transition whose associated event is true). The events associated to all the transitions starting from the current state are evaluated, and therefore the perceptions they are based on. Thus, a list of fireable transitions is built. A transition is chosen among the elements of this list, with respect to the defined policy (a random choice by default). Then, the action of the chosen transition is executed and the incoming state of the agent becomes the final state of this transition.
- If the goal is not completed, the events associated to all the transitions starting from the current state are evaluated as described above.
  - If there is at least one fireable transition, one is chosen as in the former situation, the action is executed and the incoming state is reached.
  - Otherwise, the agent executes again the current state. When the agent reaches its final state, it stops.

3.6 Consistency rules

In the SPACE model, an agent is represented by 3 models: the perceptions model, the actions model and the behaviour model. These three models are not only complementary, but also interconnected. So, consistency rules have to be defined between these models to determine if an agent is consistent. They are presented in the following. These rules have consequences on the inner structure which are also detailed.

Structure/Perceptions consistency: there is only one simple consistency rule between these two models concerning the definition of the variables : all the variables named in the perceptions (IMP and IVP) must be defined in the inner structure. The existence of this single rule comes from the obvious split of the model.

Perception/Event consistency: events must rely on perceptions defined in the perception model or on other events described previously in the actions model. To determine if an event occurs, values of perceptions or events it relies on must be specified. These values must be in the type of the perception result for a perception or in true, false for an event.

Event/Automaton consistency: all the transitions of the automaton must be labelled by an event defined in the actions model. The precondition of the action to perform must be the event associated to the transition. Moreover, when the goal associated to a state s is reached, at least one of the events associated to the transitions whose initial state is s must be true.

Automaton/Action consistency: the action A of each RMP (taking into account the answer to a request message) must belong to the actions associated to the states or to the transitions.

Actions/Inner Structure consistency: the free structure of the actions makes general consistency rules impossible to define. However, we can notice that each action must preserve the definition domain of every inner variable it may modify.

4. APPLICATION TO GRAPH COLOURING

4.1 The graph colouring problem

We describe in this part the application of the method and the model presented before to a graph colouring problem. The *general problem* is to colour the nodes of a graph with a minimal number of colours without two neighbour nodes having the same colour (LC).

The problem of graph colouring being NP-hard, algorithms looking for optimal solutions are numerous [5] but rarely useful for real-size problems. We can refer to [6, 9, 10, 11] for various methods trying to solve this problem, and more precisely to [2, 25] for ants algorithms.

The essential characteristic of our solution is that it starts with a correctly coloured graph but not optimal as far as the number of colours is concerned. For instance, a trivial initial solution consists in assigning a different colour for each node. As the time goes, our algorithm tries to suppress colours, keeping a correct coloured graph. So, the more our algorithm will work, the more pertinent the proposed solution will be.

For details about graph definitions and properties, it can be referred to [4] for example. Here are given the main ones used in the sequel. We denote $G = (N, E)$ an oriented (resp. non oriented) *graph* with N and E two sets such that elements of E are ordered (resp. unordered) couples $(u, v) \in N^2$, and $N \cap E = \emptyset$. The elements of N are *nodes*, those of E are the *edges*. Two nodes u, v of G are *neighbours* if $(u, v) \in E$. $V(u)$ will denote the set of all neighbours of u.
Let $C(u)$ the colour associated to a node u, and $C(V(u))$ the set of colours of u neighbours. The k-colouring of a graph $G = (N, E)$ is the attribution, to each node, of a colour among k such that, for each edge (u, v) of E, $C(u) \neq C(v)$. A graph is *k-colourable* if a k-colouring can be applied[1]. The smallest k such that G is k-colourable is the *chromatic number* of G and denoted $\chi(G)$.
In the following, we will consider a k-coloured graph.

For this application, we have to define two new specific notions concerning nodes. The *local chromatic number* of a node u is $lcn(u) = max\{|C|, \forall C \text{ clique of G } / u \in C\}$. The *current chromatic number* of u is $ccn(u) = |\{c(u)\} \cup \{c(v)/v \in V(u)\}|$. Then, a node u *satisfies its lcn* if and only if $lcn(u) = ccn(u)$.
The following properties are used to implement our solution.

THEOREM 1. *Let $G = (N, E)$ be a graph. For all node $u \in N$, if G is correctly coloured, then $ccn(u) \leq lcn(u)$.*

THEOREM 2. *Let $G = (N, E)$ a graph, and let $\chi(G) = n$. For each node $u \in N$, we have $lcn(u) \leq n$.*

REMARK 1. *Let us notice that, despite these two theorems, even if all the nodes of a graph satisfy their lcn, the chromatic number of the graph can not always be reached. There are also some graphs which can not be coloured such that each node satisfies its lcn.*

4.2 Application of the method

4.2.1 Global variant

The goal is to make decrease the number of colours of the nodes of a graph which is the chosen global variant. The lower bound of this variant is the chromatic number of the graph.

[1] For $k \geq 3$, decide whether a graph is k-colourable or not is NP-hard.

4.2.2 Local decomposition

The previous property is decomposed into subproblems for each node. Each node tries to change the colours of its neighbours in order to make the local variant decrease. This variant corresponds to the node *ccn* whose lower bound is the node *lcn*.

4.2.3 Agentification

The previous decomposition does not follow the locality principle. Indeed an agent should modify the colour of another agent. So, each $POSP$ consists in the colour modification of a node. The new associated local variant is the tuple of the *ccn* of the neighbours of the node. Then, each $POSP$ is assigned to an agent called a *node agent*. It can be reached by solving a set of subgoals. A subgoal consists in decreasing the *ccn* of a given neighbour. Notice that as each node agent has at least one neighbour, so its neighbours will make its *ccn* decrease.

4.2.4 Agents Behaviour

When the $POSP$ assigned to an agent is not satisfied (that is one of its neighbours has a *ccn* greater than its *lcn*), it has to choose a colour:

- existing in the graph;
- making the *ccn* of the neighbour n decrease;
- being different from the colours of its neighbours.

As a node agent can only see (and communicate with) its neighbours, to find a colour verifying the two first items enumerated above, it asks to its neighbour u the colours of its neighbours, which gives a first set $C(V(u))$, the colours set of the neighbours of u.

To verify the third point, the active agent a first asks to its neighbours their colours and constructs the set $C(V(a))$ of these colours. Then it chooses a colour among the new set $S = C(V(u)) \backslash \{C(V(a)) \cup C(a)\}$[2].

If a node agent u can not change its colour, necessarily, the set $C(V(u))$ is a member of the set $C(V(a))$. In such a case, u attacks one of its neighbours whose colour is in the set $C(V(u))$. If it is attacked, it flees, trying to take an other colour. It chooses a colour among the neighbours' ones of its neighbours, which is not a colour in its neighbours' colours.

Two other agents have to be created for coordination and implementation reasons. The topological agent creates the initial graph, node agents with their characteristics (e.g. list of neighbours, initial colours), and a drawer agent giving a graphic view of the graph updated when colours change.

4.3 Application of the model

In this part, we partially describe the node agents of the graph colouring problem presented above with the SPACE model. In this section, names of inner variables, actions, events and perceptions are respectively suffixed by *_V*, *_A*, *_E* and *_P*.

[2] The new colour must be different from the previous one, that is why $C(a)$ is removed from possible colours.

4.3.1 The behaviour model

The global behaviour of a node agent corresponds to an eco-agent one [7] which is formalised by a transducer described in [24]. The main goal of an agent is to decrease its own *ccn* as much as necessary. If it is impossible, it can attack an obstructing neighbour agent. An attacked agent can flee by changing its own colour. For instance, we detail the state of the transducer corresponding to an attack. In this state, the agent begins to evaluate if the attack is possible. If it is the case, the agent determines a target agent among its neighbours. Then, it sends an attack message to it. Finally, it analyses its mailbox and updates its goals list. Here is the description of this state in the SPACE model.

- N = "attack state";
- L = ((true;setAttackTo1_A); (true;canIAttack_A); (attacking_E;determineTarget_A); (targetExists_E;attackTarget_A); (true;readMailbox_A); (true;setupGoal_A); (true;setAttackTo0_A));
- M = void;
- Ve = {neighboursNumber_V; neighbours_V; neighboursColour_V; neighboursNeighbourColour_V};
- Vs = {neighboursAttacked_V; goals_V; attack_V};
- V = attack_V;
- π = {attacked_E > satisfied_E > unsatisfied_E};

Some of the actions or variables used above are described further in the paper.

4.3.2 The inner structure

Hereafter is given a part of the inner structure of a node agent.

- myCCN_V : integer;
- myColour_V : Colour;
- currentState_V : integer;
- neighboursNumber_V: integer;
- neighbours_V: list of Agents;
- neighboursColour_V: array of Colours indexed by neighbours numbers;
- neighboursCCN_V: array of ccn values indexed by neighbours numbers;
- neighboursNeighbourColour_V: set of Colours. This is the set $C(V(a))$ described in 4.2.4.
- goals_V: list of Goals;
- attack_V: {0; 1};
- neighboursAttacked_V: set of (Agent; Colour).
- fleeSuccess_V: {true, false} (flag indicating whether the last flee attempt succeeded).

4.3.3 The perceptions model

4.3.3.1 Information Messages Perceptions

For a node agent, we define four IMP as described below:

1. Current Chromatic Number Receipt
   - N: CCNReceipt_P
   - C: IMP
   - S: CCNMessage class with two attributes sender(*Agent*) and ccn(*int*)
   - V: {neighboursCCN_V[sender]}
   - F: neighboursCCN_V[sender]:=ccn
2. Colour Change Receipt
   - N: ColourChangeReceipt_P
   - C: IMP
   - S: ColourChangeMessage class with two attributes sender (*Agent*) and colour (*Colour*)
   - V: {neighboursColour_V[sender]; myCCN_V}
   - F: neighboursColour_V[sender]:=colour; myCCN_V := 1 + card(neighboursColour_V);
3. Neighbours Neighbour Colour Receipt
   - N: NeighboursNeighbourColourReceipt_P
   - C: IMP
   - S: NNCMessage class with two attributes sender (*Agent)* and colourSet (*Set of Colours*)
   - V: {neighboursNeighbourColour_V}
   - F: neighboursNeighbourColour_V := colourSet

4.3.3.2 Request Messages Perceptions

For each node agent, we define two RMP called *Attack Receipt* and *Neighbours Colour Request*. We only detail the first one below. An attack is taken into account only if the agent is not currently fleeing or if it has just failed to flee.

- N: **attackReceipt_P**
- C: RMP
- S: fleeRequestMessage class with one attribute sender (*agent*)
- F: (currentState_V != "fleeState" or (currentState_V == "fleeState" and fleeSuccess_V == false));
- A: flee_A (described later).

4.3.3.3 Inner Variables Perceptions

We only give one of them called *CurrentState_ P* that allows the agent to have information about its current state.

- N: **currentState_P**
- C: IVP
- V: {currentState_V}
- T: state
- F: currentState_V $\mapsto$ currentState_V (identity function).

4.3.4 The actions model

First, we need to define the three following events :

stateAttack_E is an event that is true if the current state is the attack state :

stateAttack_E : {currentState_P == attack}.

underAttack_E is an event that is true if the agent received a *fleeRequestMessage*:

underAttack_E: {attackReceipt_P == true}

neighboursNeighbourColourReceipt_E is an event that is true if the agent received a NNCMessage :

neighboursNeighbourColourReceipt_E:
{neighboursNeighbourColourReceipt_P == true}

Here are now two examples of actions relying on these events:

setAttackTo1_A:

- PRECONDITION: stateAttack_E
- METHOD: attack_V := 1;
- POSTCONDITION: attack_V == 1

Another kind of action is a response to a request message. This is for instance the case of the *flee_A* action. By this action, the agent tries to change its colour. It searches a colour among the set of the colours of the neighbours of its neighbours (*globalnnc*) and that is not a colour of its own neighbours.

flee_A:

- PRECONDITION: underAttack_E;
- METHOD:
  globalncc: *set of colours of all neighbours of neighbours*
  neighboursNumber: *number of remaining awaited answers*
  globalnnc := $\emptyset$
  broadcastMessage(NeighboursColourRequestMessage);
  neighboursNumber := card(neighbours_V);
  while (neighboursNumber $\neq$ 0) {
      neighboursNeighbourColour_V = $\emptyset$;
      while(! neighboursNeighbourColorReceipt_E);
      globalnnc = globalnnc $\cup$
          neighboursNeighbourColor_V;
  }
  globalnnc = globalnnc -
      (myColour_V $\cup$ neighboursColour_V);
  if (globalnnc $\neq \emptyset$) {
      myColour_V = choice(globalnnc);
      fleeSuccess_V = true;
      broadcastMessage(ColorChangeMessage(me, myColour_V));
  }
  else
      {fleeSuccess_V = false;}
- POSTCONDITION: true;

5. CONCLUSION AND FUTURE WORK

The method we presented in this paper helps to design a multiagent system to solve a given problem. It gives guidelines to decompose problems for agentification. The goal of the SPACE agent model is to help to specify, and then to implement resulting agents, while keeping some essential properties. Indeed, our aim is to design a model so as to be a good compromise between a purely executable agent specification and a too formal one not really implementable helping both to implement and to validate the agent behaviour. These two goals, nearly conflicting, are not both reached in other models such as Metatem [12] or Gaïa [27] for example, in which many aspects are either too or not enough formal. Specifications of agents in Metatem are hard to write because expressed in temporal logic. Moreover, no proof obligation rule is defined. In Gaïa, essentially built to give a method, many aspects are not formal enough to make a system validation possible. For instance, there is no semantics specifying how many roles taken into account by a unique agent work together. Our requirements about SPACE imply the existence of variants, accurate descriptions of perceptions, actions and events, but also consistency rules allowing to verify the inner consistency of the model when applied to a particular agent specification.

The example we give with the graph colouring problem is an academic complex case for these tools. The first outlook of this work is to provide multiple implementations of SPACE in different MAS development platforms in order to make the model easily usable. Secondly, we would try to integrate SPACE in the complete chain from the definition of a problem to an efficient MAS solving it. More precisely, the goal is to extend the method in order to produce an agent model expressed with SPACE. This step is necessary because of the richness of the model induced by its declarative aspect. This method must also provide tools allowing to exploit the different components of the SPACE model in order to check and validate agents and then the MAS. Application of the method to problems with no trivial solution should also be studied. Further steps will be to search an adaptation of these tools to other kinds of problems.

6. ADDITIONAL AUTHORS

Dr. Marianne Flouret, senior lecturer at the university of Le Havre. (LIH (EA 3219),
email: `Marianne.Flouret@univ-lehavre.fr`).

7. REFERENCES

[1] V. Jay D. Olivier C. Bertelle, M. Flouret and J.-L. Ponty. Automata with multiplicities as behaviour model in multi-agent simulations.

[2] F. Comellas. An ant algorithm for the graph colouring problem. http://citeseer.nj.nec.com/112038.html.

[3] Y. Demazeau. Voyelles, habilitation à diriger des recherches, 2001.

[4] R. Diestel. *Graph Theory.* Springer-Verlag, New-York, 2000.

[5] dimacs92. Clique and coloring problems, a brief introduction, with project ideas, 1992. ftp://dimacs.rutgers.edu/pub/challenge.

[6] Raphaël Dorne and Jim-Kao Hao. A new genetic local search algorithm for graph coloring. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 745–754, Berlin, 1998. Springer. http://citeseer.nj.nec.com/dorne98new.html.

[7] A. Drogoul. *De la simulation multi-agents à la résolution collective de problèmes : une étude de l'émergence de structure d'organisation dans les systèmes multi-agents.* PhD thesis, Univ. Paris VI, 1993.

[8] J. Ferber and O. Gutknecht. Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems, 1998.

[9] G. Ribeiro Filho. Improvements on constructive genetic approaches to graph coloring. http://citeseer.nj.nec.com/242708.html.

[10] G. Ribeiro Filho and G. Lorena. A constructive genetic algorithm for graph coloring, 1997. http://citeseer.nj.nec.com/filho97constructive.html.

[11] G. Ribeiro Filho and G. Lorena. Constructive genetic algorithm and column generation: an application to graph coloring, 2000. http://citeseer.nj.nec.com/filho00constructive.html.

[12] M. Fisher. A survey of concurrent metatem - the language and its applications. In D.M. Gabbay and H.J. Ohlbach, editors, *Temporal logic - Proceedings of the first international conference*, pages 480–505. LNAI, 1994.

[13] M. Flouret, D. Fournier, B. Mermet, and G. Simon. Formalisation des agents : du modèle conceptuel au modèle opérationnel : le modèle SPACE. Technical report, Laboratoire d'Informatique du Havre, 2003.

[14] Tony Garneau and Sylvain Delisle. Evaluation comparative d'outils et d'environnements. In *JFIADSMA*, pages 281–284. Hermes, 2002.

[15] O. Gutknecht and Jacques Ferber. La plateforme madkit et l'outil de conception sedit. In *Systèmes Multiagents : Méthodologie, technologie et expériences, JFIADSMA00*, pages 281–284. Hermes, 2000.

[16] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. Spécification structurelle, fonctionnelle et déontique d'organisations dans les sma. In *Systèmes Multiagents et Systèmes Complexes, JFIADSMA02*, pages 205–216. Hermes, 2002.

[17] Carlos Iglesias, Mercedes Garrijo, and José Gonzalez. A survey of agent-oriented methodologies. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 317–330. Springer-Verlag: Heidelberg, Germany, 1999.

[18] Bruno Mermet. Formal model of a multiagent system. In Robert Trappl, editor, *Cybernetics and Systems*, pages 653–658. Austrian Society for Cybernetics Studies, 2002.

[19] Bruno Mermet, Gaële Simon, and Marianne Flouret. A multi-agents approach for a graph colouring problem. In SCI, editor, *SCI*, 2002.

[20] H.S. Nwana, D.T. Ndumu, L.C. Lee, and J.C. Collis. Zeus : a toolkit for building distributed multi-agents systems. *Applied Artificial Intelligence Journal*, 13 (1/2):129–185, 1999.

[21] A. Rao and M. Georgeff. BDI agents from theory to practice. In *Technical note 56*. AAII, 1995.

[22] G. Rimassa, F. Bellifemine, and A. Poggi. Jade - a fipa compliant agent framework. In *PMAA'99*, pages 97–108, 1999.

[23] Arsène Sabas, Sylvain Delisle, and Mourad Badri. A comparative analysis of multiagent system development methodologies : Towards a unified approach. In Robert Trappl, editor, *Cybernetics and Systems*, pages 599–604. Austrian Society for Cybernetics Studies, 2002.

[24] Gaële Simon, Marianne Flouret, and Bruno Mermet. A methodology to solve optimisation problems with mas; application to the graph colouring problem. In *AIMSA*. LNAI, 2002.

[25] A. Vesel and J. Zerovnik. How good can ants color graphs? *Journal of computing and Information Technology - CIT*, 8:131–136, 2000. http://citeseer.nj.nec.com/443529.html.

[26] Michael Wooldridge and Nicholas R. Jennings. Pitfalls of agent-oriented development. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 385–391, New York, 9–13, 1998. ACM Press.

[27] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.