Planification for Spatially Situated Agents in Simulations

Damien Devigne Philippe Mathieu Jean-Christophe Routier
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq, France.
Email: devigne@lifl.fr,mathieu@lifl.fr,routier@lifl.fr

***Abstract*— Most often, agent-based situated simulations use reactive agents. While the use of this paradigm offers a rather natural way to build simple behaviours, it is not so easy to define complex behaviours, and these are often built ad hoc and requires precise adjustements. Moreover reactive model is not well fitted to describe deliberate group behaviours since it is more intended to emergence.**
For these reasons and in order to obtain more explicative models, we are working on proactive agent-based situated simulations. We aim at proposing a generic cognitive agent model. Such agents must build plan to achieve some goals and thus to propose a behaviour. In this paper we show why and how classical planification works must be adapted to the particular context of spatially situated agents in simulations.

***Index Terms*— agent based simulations, cognitive agents, planification, situated agents**

I. Introduction

Our work aims at producing agent-based simulations where the agents, situated in a geographical environment, are presenting rational behaviour. An application of such a work can be simulation platforms like video games are.

According to J. Laird, they constitute a "Killer Application" for human-level AI [1]. The characters involved in video games have indeed to be perceived as autonomous entities with increasing realistic behaviours. They have to be *convincing*, thus their behaviour must comply with the rational expectations of their partner or opponent human players. They also need to adapt to new situations, acquire additional abilities throughout the game, etc. In addition, team strategies are also often useful. In order to develop such kind of interactions, the agents have to make the human observer think that they behave in an "intelligent", "rational" way, as a human would behave in order to achieve its own goals.

This domain combines difficulties that can be encountered in classical AI (knowledge representation, planification), in Distributed AI (coordination of agents having most of the time different individual goals), and in Software Engineering (reusability of conceptual and software tools).

Some propositions have been done concerning agents and games [2],and most of them concern reactive agents [3]. But reactive agents, while effective in several cases, offer limited behaviours. Indeed their behaviours are "short term directed" and not "goal oriented". Their ability to perform some tasks depends on the immediate surroundings and does not result of wilful acts. Our proposition aims at offering cognitive, driven by goals, proactive agents.

The generic approach we would like to promote assumes that a single formalism can be used to design computer models either for simulating natural phenomena or for producing realistic behaviours in an artificial world in general and in games in particular. Thus from one simulation to another the cognitive behavioural engine stays the same even if the context changes and the behavioural components, we call interactions, can be (partially) reused. The main principle is to base the simulations dynamics (and the knowledge representation too) on interactions between agents: some agents can perform interactions and other agents can suffer them.

One particular point is that we want to reach some genericity in the interaction coding. The interactions code the rules of the modelised universe and, most of them, can be reused from one simulation to another. To reach this genericity we do not explicitly mention the "move actions" in the interaction. These are consequence of a particular context and then cannot be explicitly mentionned.

Since we want proactive (cognitive) agents, we provide them a behaviour's engine. It is responsible of the determination of the action the agent must (try to) perform. The chosen action is supposed to help to reach the agent objectives. To make this choice, the engine builds plan. Planification has been subject to numerous works, but we are here in a particular case because agents evolve in situated continuous environments. This situated aspect will have impact on the planification works as we will see here.

In the next section we discuss about the influences of the situated aspect on the planification and how it modifies classical approach. Then we present the principles of the situated planification algorithm used by our agents in our simulations.

II. Planification in situated contexts

There exist numerous works on planification. However while working in a situated context, this works need to be adapted. The main reason is that to perform the actions that are proposed by the plan, moves must be done by the agents in their environment.

We will present and discuss here the problems implied by the "spatially situated" aspect of the simulations and the (imperative) need to execute them rationally.

A. Monkey is not situated

In the majority of works on planning ([4], [5]) non situated problems are considered and built. Even if some appearances in the description of the subject can lead to think that the solved problem is situated, most often the moves are not planned and above all the plan is not really executed and therefore not confronted with any situated environment.

This is the case for the rather classical planification toy problem: the monkey-banana problem. This problem gives the feeling of a situated problem but the plan provided to solve it does not mention moves. Relative positions are not really considered and moves are not effectively performed, see Figure 1.

Another example could be the dinner-date problem presented in [5] to illustrate the GRAPHPLAN algorithm. In this problem one must prepare a surprise date and then the goal is to take out the garbage, to fix dinner, and to wrap a present. Here again, in the problem coding, and by way of consequence in the solution proposed by the graphplan algorithm, no moves are taken into consideration. Thus if one wants to visually simulate the resolution of this problem by a situated agent, the provided plan can not be used as it is. Indeed, in a realistic simulation, the trash or the present are located in some places that must be reached before the agent interacts with the object. The same occurs with the kitchen in which dinner is fixed.

This problem can not be easily tackled using GRAPHPLAN. As the authors say in [4], "*one main limitation of* GRAPHPLAN *is that it applies only to* STRIPS*-like domain*" and knowledge must "*be determined statically*".

Even if some works introduce conditional action schemata and adapt the planning graph expansion routine [6], [7], they do not solve the problem. First they lead to an explosion of the number of STRIPS actions and can produce as many as n^p new actions for only one action with n conditional effects, each containing p antecedent conjuncts [6]. This number is multiplied by the number of initial conditional actions. Second, GRAPHPLAN descendants require that the world being modeled is defined statically. However in simulations, one must considered open, dynamic universe and elements of the simulations can move and, for example, the locations of elements can not be statically fixed.

B. Abstract plans and execution plans

Since the main goal of our work is the simulation, we must always have in mind that any computed plan must be effectively executed in a situated environment. This introduces different levels in plans.

We will name *abstract plan* a plan where the positions (relative or absolute) of the agents are not taken into account. We will name *concrete plan* or *execution plan* the plan due to the execution of an abstract plan in a particular **situated** context. For a given abstract plan, several execution plans are possible depending on the contexts.

Now, a problem arises when the agent wants to execute the computed plan. The next action to be performed by an agent is chosen according to the abstract plan (actually, the abstract plan can offer several possible "commutative" actions). The trouble comes when this action is an interaction between him and an other agent. And it is often the case. The notions of distance and neighbourhood interfere here, since knowledge representation, and the desired realism, can require that the two agents must be "sufficiently near" for the interaction to be executed. Therefore the actor must **first** move in order to come near the target agent.

Here a difference is introduced between the abstract plan produced by a "standard" plan builder and the "real" plan due to the execution in a situated environment.

Thus, assume that, to solve a problem, we obtain the following abstract plan[1]: "$\{a_1, a_2, a_3\}$ *then* $\{a_4, a_5\}$", where the positions of the implied agents have no influence.

Even in a situated context, the existence of such a plan has a meaning. It corresponds to some generic and abstract level. This corresponds to the plan you can tell to someone to explain him how to solve a problem. For example "*to build shelves, you need planks, nails, a saw and a hammer, then cut planks at the right dimension and assemble them*". This plan remains valid in several geographical contexts. But it is different with its execution which is context dependant, for example if one must fetch nails in a near room or buy them in a store.

Moreover the execution of this plan, despite of its validity, can be impossible (at least temporarily) in a particular current context, because the door of the near room is locked or because one have not enough money to buy the nails. In this case, the plan must be extended to solve the particular contextual subproblems ("unlock door" and "fetch money" in our examples).

Actually, the situated context and the need to execute "rationnally" the abstract plan leads

- to integrate required moves into the plan,
- to extend the previously (abstract) computed plan to plan moves if required.,
- to arrange the execution of the actions according to rationality.

In the following paragraphs, we present these three points and explain why adding the spatially situated side is not just building another plan where move actions are taken into account.

C. Integrating moves.

When an abstract plan is executed, it is necessary to confront it with the geography of the environment. As we mentioned earlier, the actor agent must approach the target agents to execute the (inter)actions. But to move is an action, and therefore must be integrated into the execution plan. Thus, for each action of the abstract plan comes one move action and these move actions are then grafted on the initial plan.

In the plan $\{a_1, a_2, a_3\}$ *then* $\{a_4, a_5\}$, if we name m_i the move to reach the target of the interaction a_i, we obtain a "new" plan were move actions have been interlaced: "$\{\{m_1$ *then* $a_1\}$, $\{m_2$ *then* $a_2\}$,$\{m_3$ *then* $a_3\}\}$ *then*

[1] where the a_i are interactions, other than moves, with other agents. Groups between braces can be executed in any order, *then* mentions a required sequentiality between groups of actions

The monkey and the banana.

"*There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box that the monkey can use.*

The monkey can perform the following actions: walk on the floor, climb the box, push the box around (if he is already at it), and grasp the banana if he is standing on the box and directly underneath the banana.Can the monkey grasp the banana?"

We describe the situation by *state(M,B,O,G)* and we will assume that:

- M tells us where in the room the monkey is, possible values are *door*, *window* or *middle*,
- B tells us where in the room the box is, possible values are *door*, *window* or *middle*,
- O tells us if the monkey is on the box, possible values are *onfloor* or *onbox*,
- G tells us if the monkey has the banana, possible values are *has* or *hasnot*.

From the description, we see that the initial situation can be described as: *state(door, window, onfloor, hasnot).* The final state is *state(_,_,_,has)*.
Looking at the problem description, we see that the monkey can perform four basic actions: *walk* (walk around), *push* (push the box around), *climb* (climb on the box) and *grab* (grab the banana). Note that *walk* and *push* will be represented as two-place structures; that is: *walk(P1,P2)* will represent the action of walking from position *P1* to position *P2*.

The ability of the monkey to perform these actions (and then to change the state) is describes by the *perform* predicate: *perform(A,S1,S2)* means that performing the action A takes us from state S1 to state S2.

```
perform(grasp,
        state(middle, middle, onbox, hasnot),
        state(middle, middle, onbox, has)).
perform(climb,
        state(MP, BP, onfloor, H),
        state(MP, BP, onbox, H)).
perform(push(P1,P2),
        state(P1, P1, onfloor, H),
        state(P2, P2, onfloor, H)).
perform(walk(P1,P2),
        state(P1, BP, onfloor, H),
        state(P2, BP, onfloor, H)).
```

A solution is: *walk(door,window)* then *push(window,middle)* then *climb* then *grasp*

The chameleon and the fly.

"*There is a red Chameleon. A fly is sleeping on the ceiling. The chameleon is hungry and wants to cach the fly, but his tongue is not long enough from the floor. There is a blue box on the floor.*

The chameleon can perform the following actions: change his color, paint the box (and he changes his color simultaneously, hey he is a chameleon after all!), catch the fly if he is standing on the box and if the box is green. Can the chameleon catch the fly?"

We describe the situation by *state(C,B,O,G)* and we will assume that:

- C tells us the color of the chameleon, possible values are *red*, *blue* or *green*,
- B tells us the color of the box, possible values are *red*, *blue* or *green*,
- O tells us if the chameleon is on the box, possible values are *onfloor* or *onbox*,
- G tells us if the chameleon has the banana, possible values are *has* or *hasnot*.

From the description, we see that the initial situation can be described as: *state(red, blue, onfloor, hasnot).* The final state is *state(_,_,_,has)*.
Looking at the problem description, we see that the chameleon can perform four basic actions: *change-color* (change his color), *paint* (paint the box - and change his color), *climb* (climb on the box) and *catch* (catch the banana). Note that *change-color* and *paint* will be represented as two-place structures; that is: *change-color(C1,C2)* will represent the action of changing from color *C1* to position *C2*.
The ability of the chameleon to perform these actions (and then to change the state) is describes by the *perform* predicate: *perform(A,S1,S2)* means that performing the action A takes us from state S1 to state S2.

```
perform(catch,
        state(green, green, onbox, hasnot),
        state(green, green, onbox, has)).
perform(climb,
        state(CC, BC, onfloor, H),
        state(CC, BC, onbox, H)).
perform(paint(C1,C2),
        state(C1, C1, onfloor, H),
        state(C2, C2, onfloor, H)).
perform(change-color(C1,C2),
        state(C1, BC, onfloor, H),
        state(C2, BC, onfloor, H)).
```

A solution is: *change-color(red, blue)* then *paint(blue, green)*, then *climb*, then *catch*

Fig. 1. The monkey-banana problem is not situated (original solution and presentation (left) from `http://www.compapp.dcu.ie/~alex/LOGIC/monkey.html`). The version presented in the left side seems to be a spatially situated problem. In the right side we present the same problem, we simply replace words by others. However this second version does no more appear to be spacially situated (we keep the "climb" action for simplicity). Actually in the left version, situation have no importance since the moves are not really performed, or does not need to be taken into account since they are, by hypothesis, successful. They are more "teleportation" than moves. Therefore they simply corresponds to change of state like changing color is.

$\{\{m_4$ *then* $a_4\}, \{m_5$ *then* $a_5\}\}$" (see. Figures 2 and 3 where plans are drawn in a classical tree-like representation).

Thus we see that, simply because the abstract plan must be executed in a particular topological context that implies moves, this one must be adapt. But this raises a new problem: moves are actions and they must be planned too.

D. Planning moves

Indeed, the second perturbation introduced by moves in the plan is when to be successfully done, a move requires its own planning.

For example, this is the case when a move towards a target leads the agent to open a door, he must then plan this door opening. This action does not appear in the abstract initial plan since this door does not exist in every context, and could even already have been opened, but it is necessary for the execution of the plan in this particular context. Moreover if this door is locked, the agent has to take the appropriate key before trying to (unlock and to) open the door. This action (taking the key) must then appear in the plan, but it generates its own move towards the key too ! And so on...

Thus for a given valid abstract plan, we have a lot of different possible execution plans depending on the context of this execution, as it is illustrated in figure 4.

The resolution of the plans for each move must then be added to the one of the main abstract plan. Moreover a plan execution failure must be due to impossible moves, the validity of the initial abstract plan is not in cause in this case.

E. Arranging actions

The third effect of the situated side influences the order in which actions will be executed. Even if the abstract plan imposes no specific order in the execution of the actions that

abstract plan	:	**act on** O
situated plan	:	move to O, then **act on** O
situated plan with door	:	move to door, open door, move to O, then **act on** O
situated plan with locked door	:	move to key, take key, move to door, unlock and open door, move to O, then **act on** O

Fig. 4. Different executions plans of the same abstract plan. Execution plans depend on the context.

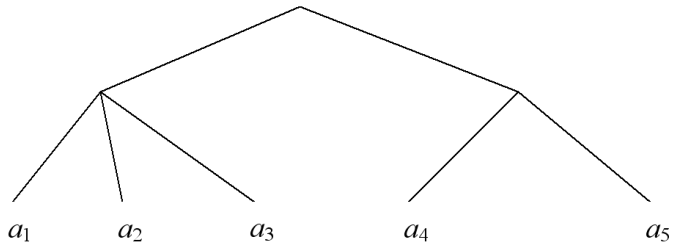

Fig. 2. An abstract plan.

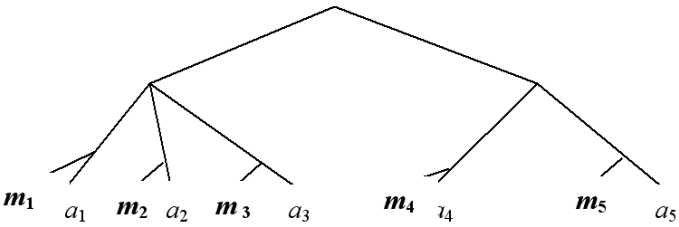

Fig. 3. One execution plan associated to the former abstract plan, the required moves m_i have been added.

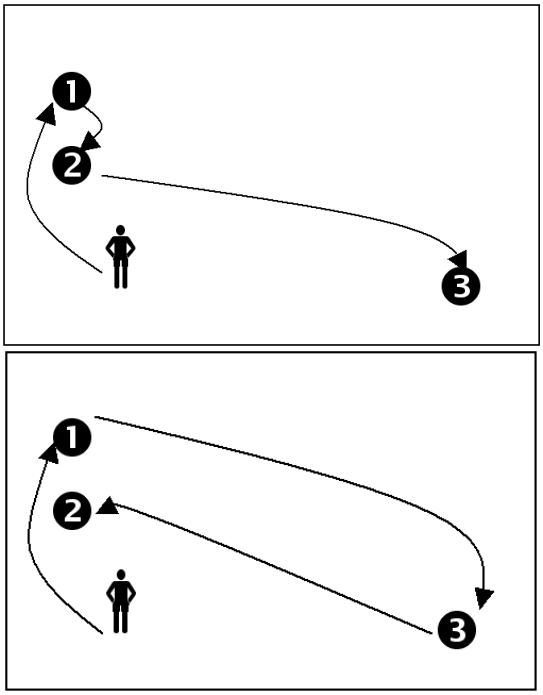

Fig. 5. Action a_i must be executed at position i with no particular order. Two executions of the same abstract plan are presented. Actions are performed in different orders and in the second, unnecessary moves are done, this will be evaluated to non rational.

compose it (because no action is precondition of another one), the execution plan can change that.

Indeed, the moves are going to be one of the more visible points while considering the rationality of behaviours and by way of consequence the realism of the simulation. If it is not realistic to always have optimal moves, it is no more admissible to have openly unnecessary extra moves. At least a local optimum of the covered distance is expected.

Let us consider an abstract plan with three actions $\{a_1, a_2, a_3\}$. No specific order is required at this abstract level and the sequences (a_1, a_2, a_3) and (a_1, a_3, a_2) are considered equivalent. But in some given situated context, they become $(m_1, a_1, m_2, a_2, m_3, a_3)$ and $(m_1, a_1, m_3, a_3, m_2, a_2)$ respectively. And, as an example, we see in the context of Figures 5 that the second execution of the same abstract plan appears to be less rational than the first one, because of the extra covered distance. And the more the positions 1 and 2 and the position 3 are distant the less the second execution appears to be reasonable.

Thus, even without trying to reach optimality, we see that moves have an influence over the order of execution of the actions because some rationality is expected.

F. Execution of the plan

A last singularity introduced by the situated side is not due to the plan building but to its execution. Indeed, once an execution plan is established, the agent must decide among all the executable actions (according to the plan), the next one to be executed. We will see in section III-D that in this case too, move actions require particular considerations...

III. PLANNING

To take this particularities into account we propose a specific planning algorithm based on backward chaining. To fulfil its goal, the (active) agent searches among all the interactions he can perform, those that can help to achieve it, and then choose one. If the conditions of this action are satisfied (according to the agent memory), the (inter)action can be fired and the plan is applied. Otherwise, the non satisfied conditions become new goals that need to be planned, and so on.

One think to keep in mind is that we want to execute the agent behaviours and we want the obtained simulation to be "rational". However rationality is of course subjective and is evaluated by a jury (the simulation observer(s)) that is external to the simulation. We will consider a behaviour as rational if a human, which would have had the same information than the agent, could have reasonably taken the same decision to perform an action to solve the (same) objective.

A. Interactions and Agents

We cannot enter here in deep details concerning the interaction and the agent models we use in our simulations. Only the notions required to the understanding of the following are presented.

The geography of the environment is a discrete map made of atomic places. Each agent is situated on a single place. Neighbourhood between places is defined by links, these links are labelled with a condition that must be satisfied by the agent who wants to use it (most of the conditions are simply *true*).

1) Interactions: Interactions are the backbone of the modelisation in our simulations. They are at the basis of the knowledge representation in the simulation since they define the actions that can be undertaken in the simulation and therefore they represent the laws of the modelised universe. Some agents (the actors) can perform interactions and others (possibily the same) can suffer them (the targets).

An interaction is described by three parts:

- a *guard*, it checks general conditions for the interaction applicability, typically it defines that to be fired an interaction requires that the distance between the target and the actor must be less than some given value.
- a *condition*, it tests the current context of execution of the interaction, it consists mainly in tests on values of target or actor properties.
- an *action*, it describes the consequence of the action, it can be a change in the state of the actor and/or of the target (ie. a change of the value of a property), and/or the activation of an environment action (like the creation of an agent).

The guard is separated from the condition since it corresponds to the knowledge due to the spatially situated aspect of the simulations. In a non situated problem, one will have only the condition and action part.

By example, to *open* an object (door, chest, window, etc.) makes it changing from *closed* state to *opened* one. The nature of the target is of no importance here, this knowledge can then be represented in a "universal" way by the interaction:

$$\texttt{open:} \begin{cases} \texttt{guard} & = & \text{“}\texttt{distance(actor, target)} = 0\text{”} \\ \texttt{condition} & = & \text{“}\texttt{target.opened} = \texttt{false}\text{”} \\ \texttt{action} & = & \text{“}\texttt{target.opened} = \texttt{true}\text{”} \end{cases}$$

The guard express that in a situated context, the actor must be near the target to open it.

We cannot detail it here but we establish something like inheritance on interactions in order to promote genricity. It is possible to particularize an interaction for some targets, thus the goal of the actor can stay the same even if he must change his plan[2].

2) Agents: We distinguish two kinds of agents: animated an inanimated agents (not to be confused with mobile/non-mobile agents). Both are defined by properties and can suffer interactions but the former can also perform interactions and have a behavioural engine.

[2]For example, for a lockable door, the `open` interaction receives one extra condition `target.isLocked = false` that must be solved (then possibly planned) even if the actor's goal remains "open the door".

The animated agents are cognitive and proactive agents. The structure structure of their "mind" is presented at Figure 6. Agents have a memory that can be seen like a "degraded environment". This one represents the knowledge base for all the information gathered by the agent concerning the environment: the topology of the environment, the other agents (their position and state). This information is used by the planification engine to determine the action that the agent must try to execute in the environment to fulfill his goal. A perception module is used to pick up information in the environment and to update the memory. Updates are performed by a separate module that has an influence on the planification engine in order to adapt the currently computed plan to the new perceived situation. This last module is a kind of "short term memory".

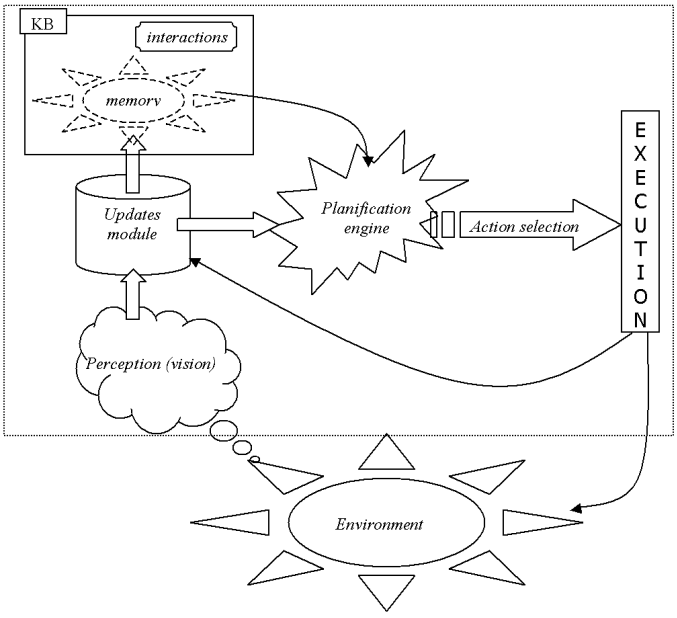

Fig. 6. Different elements of agent mind.

Animated agents have goals. The satisfaction of these goals leads the agent to behave according to a computed plan. There exist two kind of goals. First, the *interaction-goals*, they correspond to an (inter)action that the agent wants to execute. The target of this interaction can be less or more precisely given: from a named agent to any agent that can suffered the interaction, as shown in the next table:

goal	type of target
`eat(apple_12)`	a given named apple
`eat(an apple)`	any apple
`eat(*)`	any eatable (ie. "who can-suffer eat") agent

Second, the *condition-goals*, they correspond to a condition that the agent want to bring to true. For example:

`actor.energy > 100` :
"having his energy being greater than 100"

B. Planning tree

In a rather classical way, the plan produced by the backward chaining can be viewed as a tree. The nodes are made of the different goals and subgoals encountered during the resolution. Some are interaction-goals, others are condition-goals. Thus this tree is an AND-OR tree. AND-nodes correspond to condition-nodes (for condition-goals) and OR-goals to interaction-nodes (for interaction-goals).

a) Condition-nodes and interaction-nodes: A condition-node has sons only if its condition is not satisfied. These sons are interaction-nodes built from the interactions whose action part offers a way to satisfy the condition (or to approach this satisfaction, for example by increasing the energy for the above given condition-goal example). The tree leaves are the satisfied condition-nodes (ie. whose conditions are satisfied).

The interaction-node's sons are built from the conditions that can be found in the condition and guard parts of the interaction: from these, condition-nodes are built. These sons are always built. An interaction-goal is said to be satisfied when all its sons are satisfied, the associated interaction is then declared runable.

These classical and general cases being presented, we can now stay on the particular cases introduced by the situated side and more precisely the moves as discussed in paragraph II.

b) Move-nodes and exploration-nodes: For the agent, *To move* or *to explore* the environment corresponds to execution of interactions. The associated nodes must then be present in the planning tree as particular cases of interaction-nodes.

The exploration case can be reduced to the move case. To explore the agent must indeed make move towards a chosen location. The existence of the exploration-nodes are justified by the need to chose the targeted position before making the move. The agent must then apply his own exploration strategy to make his choice. Thus, in the following, we will only concentrate on the move case.

One problem is: what are the condition-nodes sons of a move-node? This problem amounts to ask what are the conditions that must be satisfied to make a move possible. To make a move a path is computed, a path is a sequence of atomic steps from atomic places to atomic places. With these places come the conditions on the links. A move is possible if these conditions are satisfied. With these conditions we create condition-nodes that become the sons of the considered move-node.

C. A backward chaining

The planning tree is built using a rather classical backward chaining algorithm.The different above mentioned kinds of nodes receive specific expansion algorithm. Every calculus is based on the memory (ie. the beliefs base) of the agent. It is in particular the case for the conditions checking or the computation of a move path. The computed plan is therefore valid according to the agent memory, but can be wrong once it faces up to the reality of the environment.

If the same (sub)goal occurs more than one time during the planning, the corresponding node is not expanded twice, it is shared by its fathers. The tree is then an oriented graph.

D. Action selection

We said that the interactions whose conditions and guards are satisfied are runable. The agent must then chose among all the runable actions the next one to be fired. This choice depends on a strategy which can be different from one agent to another.

Once the action has been chosen, the agent tries to perform it in the environment. At this moment it is necessary to check if the conditions (and guards) that the agent, according to his memory, believes they are satisfied, are actually so in the environment. If not, the agent must update his beliefs.

We said in section II that the introduction of the situated side in the planning has an influence on the order of the actions during plan execution. This influence appears during the above mentioned action selection phase: one must take into account the location of the places where the implied actions must be performed (see Figure 5).

But this is not the unique particular case. Others are more complicated.

Move actions must be assigned with a special status, at least when they correspond to moves due to the resolution of a distance guard. They can indeed not be treated in the same way that the other actions.

To explain that, let us consider the case of an interaction I having 3 conditions c_1, c_2 and c_3 and a distance guard d. The interaction-node associated to I has the 4 corresponding condition-subnodes. The expansion of the condition-node dedicated to d leads to move actions whose aim is to bring the agent to the place where the interaction I must be performed. The expansion of nodes c_i leads to interaction-subnodes in order to satisfy the conditions.

As soon as the interaction-subnodes of the c_i's are satisfied, their interaction can be fired by the agents, even if others c_j's interactions are not satisfied. That means that the satisfied interaction can be considered by the action selection phase. But, this is not the case for the move interaction-subnodes of d. Even if this move interaction-subnodes is satisfied (this is the case for example, when the agent knows a path with no condition towards the aimed place), it has no sense to consider this move action during the action selection while not all the c_i's are satisfied. This could indeed leads to the situation where the agent goes to the required place, but once he has reached it, he is unable to perform the action for which the move was made! Moreover the agent knew before he moves that this situation will occur! Thus to perform the move was not rational and therefore must not have been done. Consequently move actions must not be considered in the action selection phase while all other conditions implied in the same interactions are not satisfied (the c_i in our example). We could consider that in the subnodes of interaction-nodes (AND node), there exists two parts: the condition-nodes AND THEN the guard nodes. This illustrates a particular case implied by moves and the situated aspect.

After such a discussion, one could ask if it would not be sufficient not to make move planning (ie. no distance guard condition-node expansion) as long as all the other conditions are not satisfied. The answer is *no*. Indeed, consider the case

where, to make a move possible, some extra planning is required, something like "having that key in my drawer to unlock the door of the office I must reach". The actions (that are not moves) that correspond to this planning (something like "open the drawer and take that key" in our example) must be done as soon as possible insofar as they can be foreseen, even if all the other conditions of the interactions the agent wants to perform in the mentioned office are not yet satisfied.

As we can see, move-actions subtrees must be treated in a special way and this case does not occur in non situated context or when the rationality of the executed behaviour (plan) is not considered.

E. A partial replanning

The agent evolves in a dynamic and non monotonic environment. He can then be brought to adapt a computed plan according to new perceived information. These information can be of several types, here are the main ones:

- *a new information*: the agent learns that a so far unknown information exists. It is the case when he sees a new place or meets an agent for the first time, gets a new goal, discovers a condition on a path of the graph, etc.
- *a modification of an existing information*: it concerns mainly modifications about the state of known agents, a property value change or a position change. The position change information covers three situations:

known→known:	we thought agent at a position and we see him at another
known→unknown:	we thought agent at a viewed position and he is not there
unknown→known:	we did not know where the agent were and we see it

- *an action has been performed*: this information is used to take into account the modifications implied by an interaction performed by the agent.

It is the new information management module that is in charge of forwarding these information to the planning engine.

However, a new information concerns only a portion (even none) of the planning tree. Therefore, it is neither reasonable, nor efficient, to rebuild a new plan for every new information. Indeed, even if it is established that, in theory, no efficiency gain can be guaranteed while using plan reuse rather than new plan generation ([8]), in practice improvements can be expected. Indeed, our context of dynamical simulations corresponds to the case where the agent perceives very frequently slight changes in his knowledge base. In particular, this is due to the fact that the agent engine use uncertain information: the planification is based on the information that are in the memory, but, since simulations occur in open dynamic environments, this information are non monotonic, and the built plans are correct with respect to the memory of the agent, but can be wrong once confronted with the real environment, at the execution step. Therefore only partial and local adaptations can be expected in most of cases. Our experiments confirm that.

We do not have enough space here to detail this partial replanning. In broad lines, our approach is to top-down propagate new information, from root to leaves in the planning tree. The information determines which of the nodes are concerned by it and only the concerned nodes are updated (collapsed, re-expanded, addition or removal of subnodes, etc.). Thus information that affect the graph topology or agent positions can have an impact on move-nodes but not on others. In a similar way, a new met agent can affect a condition-nodes if this agent can be the target of an interaction that helps to solve the condition. In this case a new interaction-subnode must be added.

IV. Conclusion

In most of cases agent-based situated simulations use reactive agents. We propose to use proactive/cogntive agents. This implies to provide our agents with a planification engine. However since simulations occurs in a situated context, a particular approach of the planification is required. Indeed, plans must be executed to perform the simulation. In a situated world, this implies that the actor must moves to reach the location where an action must be fired. This need to integrate moves leads to adapt the computed abstract plan. We have discussed the problems implied by this move's integration. We propose a planification algorithm for agent in a situated context. A computed plan is partially rebuilt when agent detects new information.

We have implemented this algorithm and our cognitive agent model in a situated simulation framework. This results in about 150 java classes for implementing the environment, agent and interaction models and another 250 classes for a graphical platform that provides GUI tools for creating interactions, agent classes and instances, environment map and executing simulations. This framework has helped us to lead experiments to illustrate and to validate our propositions.

We are currently working on adapting our models to build simulations with teams of agents. With reactive systems collaborations between agents comes from emergence. With our proactive approach we want to settle deliberate cooperation between agents that are members of a team.

References

[1] J. E. Laird and M. van Lent, "Human-level AI's Killer Application: Interactive Computer Games," 2000.

[2] A. Nareyek, "Intelligent agents for computer games," in *Computers and Games, Second International Conference, CG 2000. LNCS 2063.*, 2000, pp. 414–422. [Online]. Available: citeseer.nj.nec.com/nareyek00intelligent.html

[3] ——, "Specification and development of reactive systems," in *1998 AIPS Workshop*. Menlo Park California: AAAI Press, 1998, pp. 7–14.

[4] A. Blum and M. Furst, "Fast planning through planning graph analysis," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1995, pp. 1636–1642. [Online]. Available: citeseer.nj.nec.com/blum95fast.html

[5] D. S. Weld, "Recent advances in AI planning," *AI Magazine*, vol. 20, no. 2, pp. 93–123, 1999. [Online]. Available: citeseer.nj.nec.com/article/weld99recent.html

[6] B. Gazen and C. Knoblock, "Combining the expressivity of ucpop with the efficiency of graphplan," in *Proceedings of the Fourth European Conference on Planning*. Springer-Verla, 1997.

[7] C. Anderson, D. Smith, and D. Weld, "Conditional effects in graphplan," in *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*. AAAI Press, 1998, pp. 44–53.

[8] B. Nebel and J. Koehler, "Plan modification versus plan generation: A complexity-theoretic perspective," in *Proceedings of of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, 1993, pp. 1436–1441.