APPROCHE INTER-ACTIVE DE LA RESOLUTION DE PROBLEMES: L'ECO-RESOLUTION

DELAYE C. [1-2], FERBER J. [2], JACOPIN E. [2]

RESUME

L'éco-résolution offre une nouvelle manière d'aborder la résolution de problèmes. Alors que les techniques classiques tentent de résoudre les problèmes globalement, en effectuant une exploration de l'espace des états gouvernée par des heuristiques, l'éco-résolution est fondée sur un mécanisme d'interactions entre agents, la résolution étant obtenue par un ensemble de satisfactions locales de ces agents. Ses principes reposent sur la définition d'un ensemble de comportements spécifiques appartenant à plusieurs "espèces" d'agents (réactions de satisfaction, de fuite ou d'aggression).

Le système ECO 1 se compose de deux parties: d'une part le noyau qui décrit l'interaction de ces comportements à partir d'une définition abstraite de ces agents, cette partie étant indépendante de toute application, et d'autre part, la définition d'un ensemble d'espèces d'éco-agents, dépendants du domaine, qui héritent des propriétés de ces agents abstraits, et particularisent le modèle pour une application donnée.

AN INTERACTIVE APPROACH TO PROBLEM-SOLVING : ECO-PROBLEM SOLVING

1. INTRODUCTION

The classical multiagent paradigm involves the decomposition of a problem in order for a population of agents to elaborate a solution and to carry out this solution. We address the problem of

[1] ONERA, DMI/IA, 29 Av. Division Leclerc, BP 72, 93322 Chatillon Cedex
[2] LAFORIA, Université Paris 6, T46-00, 4 Place Jussieu, 75252 Paris Cedex 05

constructing the solution with a population of agents [14]. This is Distributed Problem Solving.

Our agents are actor-based and follow Gul Agha's model of continuations [1]. They are integrated in an eco-system [6,7] where they possess very simple behaviors. But our aim is neither the simulation of an environment such as the prey-predator [5] nor the simulation of a behavior [10]. Our approach also differs from connectionism; our agents are not statically linked together and their behaviors are independant one another. Moreover, their actions are not based on stochastic functions [13]. Also, our system differs from the knowledge-based agent approach.

Our system is twofold: (1) a domain independant kernel where the eco-behaviors are described, (2) a domain dependant application where the domain actions are coded. We used it to solve classical AI problems (e.g. blocks world, hanoï towers, n-puzzle, n queens, etc...) [4,8]; in the goal of solving problems involving agents' action selection we propose to extend our model. For this purpose, we choose Roach's non linear problem [11,16] where a robot faces an action selection problem. Section 2 describes the underlying model for our eco-agents, along with Roach's non linear problem. A formal study of an eco-agent is proposed and the completeness of our system is presented. Then, section 3 presents thr runtime for Roach's problem.

Our system makes wide use of continuations. Consequently, agent's actions are serialized through the process of continuations. In order to solve Roach's non linear problem, we need to extend the notion of continuation in an actor-based system. The limits of the continuation model are expressed in section 4, and solutions are provided in section 5.

2.THE ECO-AGENT MODEL

This section presents the model of our very simple agents. Our agent are actor-based. We first present this model and describe agents knowledge and behaviors. Then, we modelize our agents and make a brief study of the completeness of a system using such agents. All this section is presented with the support of a problem whose solution is presented in the last subsection. Figure 1 represents this problem. This problem is called Roach's non linear problem. We chose this problem because it has been newly analysed as a non linear problem [11], it is not an usual blocks world problem and although it is simple, it provides a good workbench for our purpose. In this problem, a robot is at place 1 in the initial situation and must reach place 4 near the sink. The door is closed in both the initial and final situation.

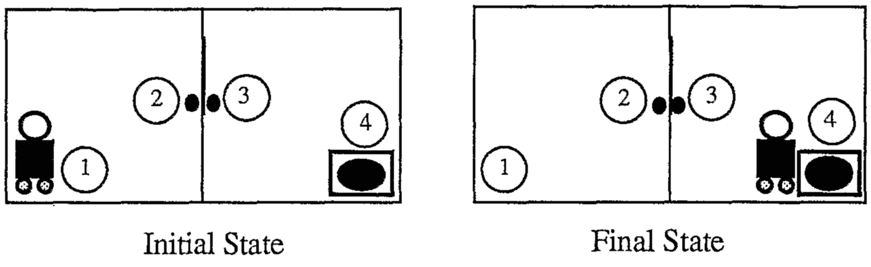

Figure 1: Roach's non linear problem

2.1 Knowledge and behaviors

In this section, we describe the model chosen for our agent. An eco-agent is actor-based [1]. It has some local knowledge of its own environment and has behaviors to modify its local knowledge.
Local knowledge. Independently of the domain, an agent knows:

1. Its satisfaction state; it is true when the agent has reached its goal, otherwise it is false and the agent is seeking for its satisfaction.

2. Its dependancies. Dependancies are agents. We call *master* an owner of some dependancies, and *slave* the agents that are the dependencies. An agent which is a dependency is a master for its own dependencies, and so forth. Slaves will reach their satisfaction state only *after* their master's one has been reached. Differences between the initial situation and the final situation define the master-slave relationship. For instance, in the problem described in Figure 1, the robot has to change its position from place 1 to place 4, near the sink. This makes the robot as the slave of the sink.

3. Its jailers. Jailers are agents that prevent other agents from acting. For instance, in the case of Figure 1, the door prevents the robot from reaching the sink, therefore, the door is a jailer for the robot. As soon as the robot will have open the door, the door will not be a jailer any longer.

Depending on the domain, an agent may have more knowledge about its environment. For instance, the robot also knows its position in the environment.
Behaviors. Agent's behaviors are independent of the application domain. Three distinct behaviors are available for an eco-agent:

1. The will to be satisfied. This will corresponds to the description of the agent's goal in the final state of a problem. In our example, the robot must reach the sink. Consequently, it get the will to be near the sink. An action (reaching its satisfaction) is associated with the will to be satisfied.

2. The will to be free. The idea is that the agent must be free before any acting. Therefore, the agent has the wil to be free each time it must act; i.e. before doing its satisfaction, and before fleeing.

3. The obligation to flee. If an agent prevents another agent from acting, then this agent must flee. For instance, the door prevents the robot from reaching its satisfaction state. So, the door must flee.

2.2 The finite state automaton

In this section we propose to formalize an eco-agent as a finite state automaton. This formalization is easily obtained. First, it is easy to derive the internal state of an eco-agent from the behaviors it can have. This internal state is a boolean triple (s,f,l) where s is the satisfaction state, f is the fleeing state and l the liberty state. Providing that an agent cannot be both in the state of fleeing and in the state of satisfaction, triples (1,1,x) where x is either 0 or 1, cannot exist. The starting state is (0,0,0) and the final states are all the possible states. Each behavior is an action on the internal state; application cases are defined through a transition function whose diagram is represented in Figure 2.

The finite state automaton brings up comparison with other works. First, it is clear that this model describes situated actions [17], but also, it is actor-based and then is different from Agre's system, Pengi [2].

2.3 Completeness

We present a brief study of the completeness of a system having the properties described in the previous subsections.

In classical planning systems [3,19], the final situation is described through a set of propositions. These propositions are asserted by the postconditions of a course of actions. This course of actions reaches the final situation from the initial situation: it solves the problem. In eco-problem solving, the final situation is described through a set of satisfaction. Hence, agents must interact in order to reach the satisfactions corresponding to the description of the final situation.

A truth criterion for eco-problem solving. *A satisfaction is asserted in the final situation (Sf) if it is asserted in the final situation or if it has been asserted in a previous situation (Sp) and there is no situation (Sb) between situation (Sf) and situation (Sp) such that the agent has fled. For each situation (Sb) where a satisfied agent must flee there must be a situation (Sa) between situation (Sb) and situation (Sf) such that the satisfaction of the fleeing agent is* *asserted. For each situation where an agent receives the obligation to flee and is able to reach its satisfaction state, then this agent will reach its satisfaction state.*

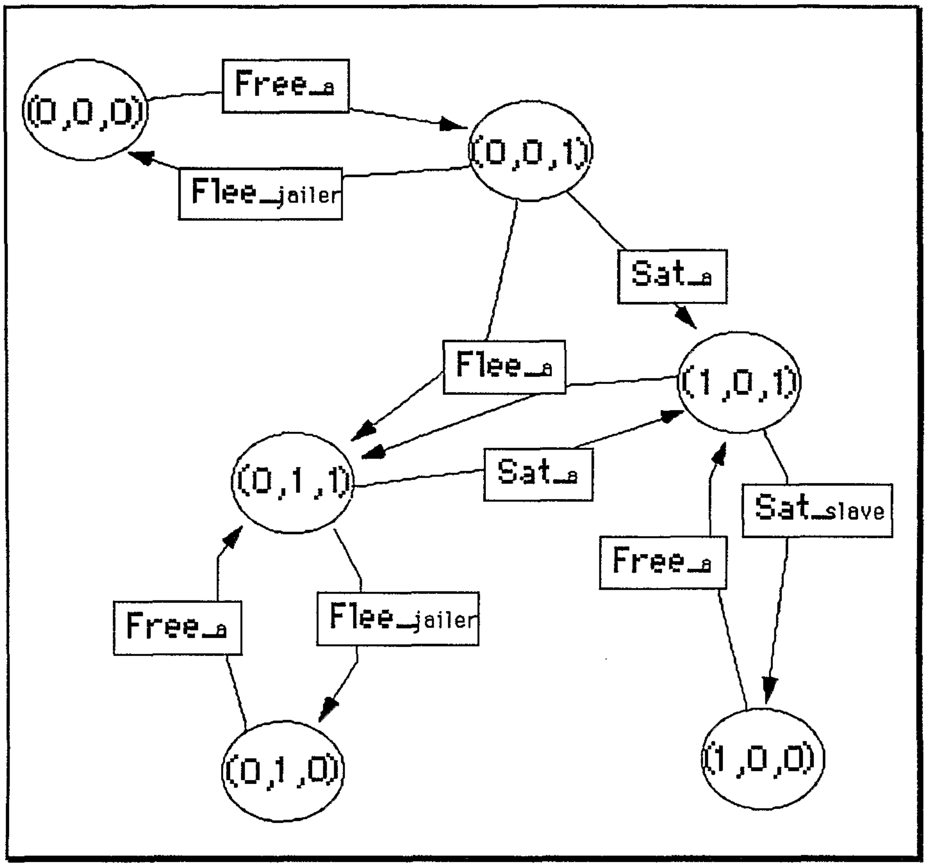

Figure 2: The transition diagram of the finite state automaton

This truth criterion clearly descends from Chapman's [3]; using his terminology, situation Sp is an *establisher*, situation Sb is a *clobberer* and situation Sa is a *white knight*. But our modal truth criterion is adapted to a multiagent environment. First, this criterion does not talk about propositions (i.e. post-conditions) but about states determined by agent's behaviours (i.e. the satisfaction state and the fleeing state). Second, agents are acting from local interactions and no global data bases (such as the usual add or delete lists) about the environment are used. Hence a situation Sa (Declobbering by white knight in Chapman's terminology) is easy to create: an agent that should be satisfied in situation Sf receives the will to get satisfaction each time it flees, although Chapman's TWEAK [3] and wilkins's SIPE [19] do not use a white knight.

3.THE RUNTIME

Now, we are going to describe the solution found by our system. The robot has to go to place 4, near the sink, and the door must be closed. First the robot tries directly to go near the sink. But the door is closed. So the robot tells the door to open; as a consequence, the door asks the robot to move near it in order to open it. Thus, the robot moves near the door and open it. As soon the door is open, the robot is free to reach its satisfaction (i.e. to be in place 4, near the sink); so the robot moves near the sink, leaving the door open. There, the door calls back the robot so as to be closed. Finally, after moving back and closing the door, the robot is free to return to the sink.

The final state is reached. But the robot did make a pretty useless return trip to the door. The problem appears as soon as the robot is free, when the robot is in place 2 and the door open. There, the problem is due to the continuation process.

We can notice that using the idea of serializing the goals [12] (going from place 1 to place 2, then to place 3 then to place 4) leads to the good planning of the tasks. But it is an undesirable solution because it also needs a meta-control to find the good serialization of tasks. So the problem would only be displaced from task-planning to control-planning.

4.THE LIMITS

The purpose of this section is to explain why the robot makes an undesirable return to the door and then acts again towards its satisfaction, near the sink. We will explain the limits of this approach and the different solutions in the following subsections.

4.1 Unneeded sequentialization with continuations

This undesirable return is the result of the programmation with continuations. Indeed, each new continuation is considered only when the actor twhich owns it has finished its current computation. So, if we examine precisely the trace of the execution of the program, we see that there is very few active actors but that each of them handles a long queue of continuations.

As in our system each continuation may correspond to the execution of one task, it means that we can't change the planning of tasks as soon as they appear in a queue of continuations. As a consequence, a queue of continuations reflects the scheduling of a succession of tasks at the instant of its creation; henceforth the scheduling of the tasks in the queue is fixed.

As a result, the satisfaction of the robot to reach the sink is in the queue of continuations. The satisfaction of the door is only taken in charge when the robot has first satisfied its goal to be near the sink as it is specified in the continuations. There is no way to avoid it even if we put priorities between tasks like "always execute first the tasks with the door as receiver" because it is impossible to break the queue of continuations.

It is important to note that this "continuation problem" seems to be one of the major drawback of programming with actors.

4.2 Task versus eco-action duration

We must now focus on another problem. In the current implementation, we have made no difference between behaviors of eco-agents and real physical execution of these behaviors in the world of the robot. In fact, going from one location to another is not an **atomic** task: it may take a long time for the robot to move, conversely, the intention to move is instantaneous [9]. But in our system, consuming one step of any queue of continuations takes exactly the same time whatever action (e.g. moving or getting the intention to move) it recovers.

In our example, the "DoGoal" of the door can't be consumed before the satisfaction of the robot. We must insist that even if we use real-time slice for our scheduling of continuations instead of a round-robin, the result would not change. In this case, the time to consume a continuation would be proportional to the time of computing its code. It would have no relationship with the duration of the execution of the task in the real world.

This also leads the robot to fail to immediately close the door: the first goal of the robot is to get near the sink, so it does it before any other consideration. As soon as the door is open, the robot is free so it satisfies itself.

5.SOLUTIONS

In order to solve our problem we want the robot to have two simultaneous abilities: (1) to go from one point to another; (2) to care of all messages that it receives. An ideal solution would be to make an agent that contains two parts: one for displacement and one for listening. The listening part would interrupt the work of displacement as soon as a more important task must be realized. The importance of a task is associated with its temporal cost in the context of the global succession of tasks.

In order to avoid an increase of complexity, we have implemented a simpler system that do not use continuous parallelism of the two tasks but is an incremental version where an agent reads its mailbox at some interval.

5.1 Incremental Actions

All the actions occuring in the real world will be incremental in order to allow each agent to read its mailbox and act in the same time. It results in similar implementations for the eco-actions interacting with the real world: DoFlee and DoSatisfaction. In the next paragraphs we will only develop the code of DoSatisfaction but the code of DoFlee is similar.

5.1.1 An incremental DoSatisfaction

We now introduce a new DoSatisfaction that moves the robot for only a small distance instead of realizing an instantaneous satisfaction (going from place 3 to place 4 at the speed of changing an actor's field). The new DoSatisfaction calls itself recursively until the robot has reached the sink. At every cycle (when a new DoSatisfaction is executed) the robot reads its mailbox. When this one is not empty, it takes the messages into account.

For the implementation, it results in a DoSatisfaction(agent's position in x,agent's position in y) that sends the message DoSatisfaction(agent's position in x + dx, agent's position in y + dy) unless there are messages in the mailbox; dx and dy are given by a extremely simple path planner which calculates the next increment towards the goal. In this case, the robot executes these messages---if needed---and then sends the normal continuation DoSatisfaction(x+dx,y+dy) as soon as all the messages received have been taken into account. Here are the outlines of functions **Satisfied?**, **IncrementSatisfaction** and **DoSatisfaction**.

```
(function Satisfied? (self)
     (return (Is self's goal reached ?)))

(function IncrementSatisfaction (self)
     (Let the current self's position be nearer to
     self's goal position))

(function DoSatisfaction (self ?constraint ?cont)
     (If self is Satisfied? then

(Carry out the current continuation ?cont)
     Else
     (When the mailbox contains a message, take care of
     it))
     (IncrementSatisfaction self)
     (DoSatisfaction self ?constraint ?cont))))
```

We have introduced a local mailbox. This mailbox may be a common one like in blackboards but it is more useful to use local mailbox for each agents. By this way, eco-agents have reached real agents status.

The solving of the precedent example is now biased in the sense that only one message may be in the mailbox (the need of the door to be closed). In real problems, multiple messages may be sent and the order in which they are executed is a real task planning.

Another problem is that it may be important to plan all he tasks to go to the sink. An example is the robot+sink+key problem that is identical to Roach's except that the robot needs a key to close the door. Here, we have to plan the execution of the task to search the key simultaneously with using the fact that we will further have to reach the sink.

5.2 A step forward dynamical planning

In a more general problem solver, we must add the goal of going from the position of the robot to the sink to the content of the mailbox. But as soon as we allow this, we need a more powerful planning to know how to merge the tasks in the mailbox with the task of going to the sink. Here is an outline of a new function **DoSatisfaction** that handles the problem:

```
(If self is Satisfied? then
    Carry out the current continuation ?cont)
    Else
    (IncrementSatisfaction self)
    (When the mailbox contains a message
    (add (DoSatisfaction self ?constraint ?cont)
    to the mailbox)
    (take care of mailbox))))
```

5.3 Tasks as eco-agents

Reorganizing the tasks in the mailbox do not seem to be of a real interest because it is only an other rewriting of the same old problems. We will see now that we may consider tasks not only as first-class objects but also as eco-agents (we have shown, in [8], that in blocks world, blocks are eco-agents).

5.3.1 Plunging the tasks in blocks world

Figure 3 illustrates our description.We may consider that the constrains between tasks are not very different from clobberers in the world of blocks; i.e. blocks on top of others.

Also, the temporal constraints between tasks are similar to the spatial constrains in blocks world. Consequently, we propose to introduce the task of going from one place to another as a block. Thus, taking a distance-block off a tower is similar for the robot to movie along the same distance.

On moving from one place to another, the robot needs to suppress the corresponding distance in the block world representation. The loci are also represented by a block. When a locus-block is on top of a tower, it indicates that the robot is on the locus corresponding to the block. The task of opening and closing the door are indicated by blocks that dynamically goes on the top of the tower, the robot cannot do anything without suppressing this block (executing the corresponding task).

Is is very important to note that this technique is nothing else than composing a "big" eco-agent with "smaller" eco-agents that handle the tasks of the "big" eco-agent.

6. CONCLUSION

6.1 Results

In this (extended) abstract, we have presented a multiagent model involving the notion of eco-agent. Its description and its formalization (finite state automaton, completeness) has been studied. Using the example of Robot and Sink, we have shown that a actor-based system have some limitations that we have solved by two means: incremental actions and tasks as eco-agents in the blocks world.

6.2 Perspectives

We have not pointed out the fact that our blocks world may be a dynamic one. Therefore, the blocks may be pushed dynamically on the top of the tower. It means that we may consider situations where the tasks of an agent may dynamically be changed. In this representation, the breakdown of a tower of blocks means a total replanning. In order to exploit this caracteristic we are currently implementing the Misachieving Baby example [15] that leads us to a more dynamical problem where tasks must constantly be reorganized from the beginning.

Another extension will be to give new dimensions to blocks world: the weight of a block may be compared to the temporal cost of a task. We may also consider more elaborated constrains between tasks using geometric blocks that may interleave together.

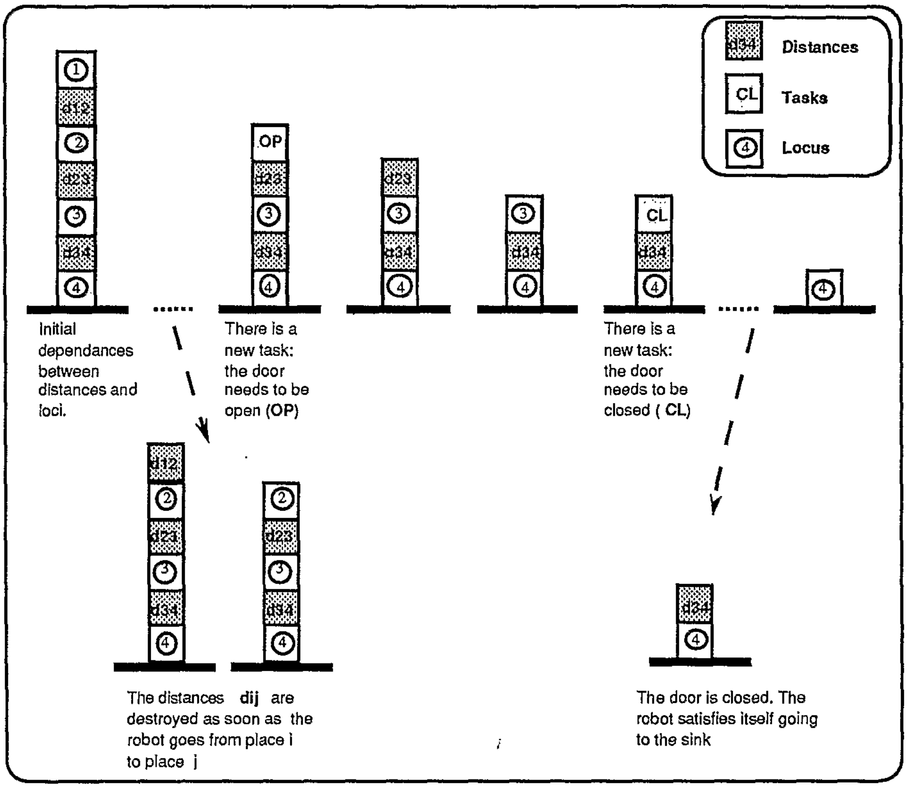
Distances
Tasks
Locus
Initial dependances between distances and loci.
There is a new task: the door needs to be open (OP)
There is a new task: the door needs to be closed (CL)
The distances dij are destroyed as soon as the robot goes from place i to place j
The door is closed. The robot satisfies itself going to the sink

Figure 3 : The robot problem revisited through blocks world

REFERENCES

[1] Gul Agha, *Actors - A model of Concurrent Computation for Distributed Systems*, MIT Press (1986).

[2] Philip Agre & David Chapman, *Pengi: An implementation of the theory of activity*, in: Proceedings of AAAI-87 (1987), pp. 268-272.

[3] David Chapman, *Planning for Conjunctive Goals*, Artificial Intelligence 32 (1987), pp. 333-377.

[4] Alexis Drogoul & Christophe Dubreuil, *Classical AI problems and Eco-problem solving*, LAFORIA working paper, LAFORIA 1990.

[5] E. Durfee, T. Montgomery, *MICE: A Flexible testbed for intelligent Coordination Experiments*, Proceedings of the 9th Workshop on Distributed Artificial Intelligence (1989), pp. 25-40.

[6] Jacques Ferber, *Objets et agents: Une étude des structures de représentation et de communication en Intelligence Artificielle*, Thèse de Doctorat d'état, Université Pierre et Marie Curie, 492 pages, June 1989. (In french).

[7] Jacques Ferber, *Eco-Problem Solving: How to solve a problem by interactions*, in: Proceedings of the 9th Workshop on Distributed Artificial Intelligence (1989), pp. 113-128.

[8] Jacques Ferber & Eric Jacopin, *A Multi Agent Satisfaction Planner for Building Plans as Side Effects*, LAFORIA Report 07/90, February 1990.

[9] Jean Marie Hoc, *Psychologie cognitive de la planification*, Presses Universitaires de Grenoble (1987).

[10] Bernardo Huberman, *The ecology of computation*, Elsevier Sciences Publication, North Holland (1988).

[11] David Joslin & John Roach, *A Theoritical Analysis of Conjunctive Goal Problems*, Artificial Intelligence **41** (1989/90), pp 97-106.

[12] Richard E. Korf, *Planning as Search: A Quantative Approach*, Artificial Intelligence **33** (1987), pp 65-88.

[13] Patti Maes, *The dynamics of Action Selection*, AI memo 89-09, AI LAB, Vrije Universiteit Brussel (1989).

[14] Marvin Minsky, *The Society of Mind*, Basic Books (1986).

[15] Marcel Schoppers, *Universal plans for reactive robots in unpredictable environment*, in: Proceeding of IJCAI-87 (1987), pp. 1039-1046.

[16] Laurent Siklossy & John Dreussi, *A Hierarchy driven robot planner which generates its own procedures*, in: Proceedings of the 3rd IJCAI, Stanford, CA (1973), pp 423-430.

[17] Lucy A. Suchman, *Plans and situated Actions, The problem of human-machine communication*, Cambridge University Press (1987).

[18] Luc Steels, AI memo 89-07, AI LAB, Vrije Universiteit Brussel (1989).

[19] D. Wilkins, *Practical Planning - Extending the classical AI paradigm*, Morgan Kaufman (1988).