How Situated Agents can Learn to Cooperate by Monitoring their Neighbors' Satisfaction

Jérôme Chapelle[1] and **Olivier Simonin**[1] and **Jacques Ferber**[1]

Abstract. This paper addresses the problem of cooperation between learning situated agents. We present an agent's architecture based on a satisfaction measure that ensures altruistic behaviors in the system. Initially these cooperative behaviors are obtained by reaction to local signals emitted by the agents following their satisfaction. Then, we introduce into this architecture a reinforcement learning module in order to improve individual and collective behaviors. The satisfaction model and the local signals are used to define a compact representation of agents' interactions and to compute the rewards of the behaviors. Thus agents learn to select behaviors that are well adapted to their neighbor's activities. Finally, simulations of heterogeneous robots working on a foraging problem demonstrate the interest of the approach.

1. INTRODUCTION

Over the last few years, learning in distributed systems has become an important field of research. Within this context our work concerns situated agents (autonomous entities that evolves in a real or simulated physical environment) that have to cooperate to achieve global tasks.

The behavior based approach is well suited to design situated agents. This kind of architecture allows defining agents that are able to evolve in dynamic and partially unknown environments (for example [1][2][3][4]).

Cooperation between situated agents can be obtained directly (i.e. by direct communication [3][5]) or indirectly (i.e. by auto-organization and agents-environment interactions [6]). Usually, agents architectures use high level direct communications. However, applications in real conditions (as in mobile robotics) induce failures and complexity.

To avoid these problems, we suggest an architecture using simple signal emissions (local diffusion of values without acknowledgement). These signals are used to manage cooperation and to define a reinforcement learning module.

In previous works a cooperative agent architecture using signals of agent satisfaction was defined [7][8]. In this approach, cooperation is based on a simple reaction to these signals (see details in section 2). This architecture is able to solve some collective problems of foraging tasks [7][8] and spatial conflicts resolution (with real and simulated robots) [4].

However, these agents are not able to remember which behavior is the best adapted to any given cooperative situation. This paper presents the integration of a reinforcement learning module using these signals in order to improve cooperative behaviors.

The agent learns, for a given situation, which behavior satisfies the majority of its neighbors. This kind of process is also known as Social Learning [9].

This paper is organized as follows. Section 2 introduces the agent satisfaction measure and describes the satisfaction-altruism model. Section 3 presents the integration of a reinforcement learning module in this architecture. Then in section 4 the model is evaluated by its application to a classic collective problem of foraging. Some results of simulations show the improvement of agents cooperative behaviors. Finally we conclude in section 5 by a discussion of these results and by presenting our future work.

[1] LIRMM UMR 5506, Université Montpellier II - CNRS, 161 rue Ada 34392 Montpellier Cedex 5, France

2. WHAT IS AGENT'S SATISFACTION ?

In order to design and study cooperative agents two generic agent's satisfaction measures have been defined [7]. The first one, the personal satisfaction, evaluates the progression of the agent's actions. The second one, the interactive satisfaction, evaluates the effect of the neighbors' actions on the agent's task.

We consider that each agent a_i holds a set of functions F_i, also called behaviors, which are useful to achieve some tasks. These functions may be released by the perception of environmental stimuli or conditions to act.

Each agent's function $\mathbf{f_i}$ is defined by

- its conditions of releasing : **cond(s)® boolean**
  - (where s is information from sensors)
- a current trigger weight : **W(t) ®[0,1]**
- a set of processes : to drive effectors
- a satisfaction measure when f_i is performed: **P Î [-1,1]**

2.1 Personal Satisfaction

The personal satisfaction is a value continually computed by each agent following its current actions.

Definition: for an agent a_i who performs a function f_k, let $P(t) \in \mathbb{R}$ be the value of personal satisfaction at time t, computed as follow:

- $P(t) = P(t-\Delta t) + v$,
- $v \in \mathbb{R}$ is a measure of the progress of f_k during Δt ($|v| \ll P_{max}$)
- $\forall t \geq 0$, $|P(t)| \leq P_{max}$, P_{max} is a constant of $\mathbb{R}^+$
- the initial value $P(t_0) = W(t_0)$.

Definition the progression of satisfaction v of the function f_k during Δt is computed as follows:

$$v = \begin{cases} m & \text{progress towards } f_k \text{ goal} \\ n & \text{regress forwards } f_k \text{ goal} \\ f & \text{immobilization of agent action} \end{cases} \quad (1)$$

with $-P_{max} \ll f < n < 0 \leq m \ll P_{max}$

This measure of v is an extension of the progress estimator functions proposed by M.J. Matarić [2]. We introduce a negative reward when the agent is immobilized. In particular, the penalty value f is greater than any negative reward of regression (f < n < 0).

This satisfaction model allows agents to control the persistence of their function execution (see figure 1). This behavioral principle may be abstracted as follows :

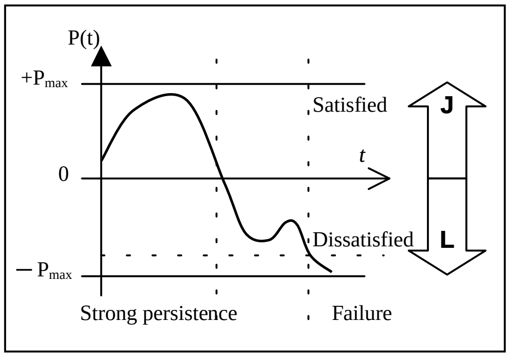

Figure 1 – Illustration of the Personal Satisfaction evolution.

- the impossibility to achieve a task induces a quick drop of the personal satisfaction which stops the current function,
- the regression of the current function induces a slow drop of satisfaction which gives the agent time to persist (or to find a new way) in doing its task.
- the progression of the current function gives a high positive satisfaction value that increases the motivation of the agent to continue its actions.

2.2 Interactive Satisfaction, Signals and Altruism

The interactive satisfaction is a set of measures of interactive situations: level of hindrance, need for help or possibility to share resources (the definition of these measures or perceptions depends on the considered application).

To manage these situations, agents can emit local signals (cf. fig. 2). They are simple relative values, noted I(t), defined in $[-P_{max}, P_{max}]$:

- a positive value to attract the neighbors,
- a negative value to repulse them.

In order to obtain cooperative behaviors, a mechanism of reaction to these signals has been defined, called the altruism behavior [7]. As agents are situated in physical environments, this altruistic reaction is defined by the computation of a movement.

An agent decides to be altruist by comparing its personal satisfaction (P) and the greater external signal intensity $|I_m|$. The agent becomes altruist when $|I_m|>P$. To satisfy the request (i.e. I_m signal) it moves by applying the altruism vector deriving from a signed potential field. For an agent B that receives a signal I_A of an agent A, the altruism vector is computed as:

$$\overrightarrow{F_{B/A}}(t) = k.sign(I_A(t)).\frac{|I_A(t)|}{\left\|\overrightarrow{AB}\right\|^{(n)}}.\overrightarrow{BA} \qquad (2)$$

Thus, the goal of the agent may become this vector which is combined with obstacle avoidance vectors. Figure 2 illustrates the complete model.

If the agent doesn't choose the altruistic reaction, it only continues to perform its current function if the weights of activable functions are less strong than P(t). Elsewhere, it selects the behavior with the highest weight W(t) (see [7][8] for more details on the satisfaction-altruism architecture).

The next section presents how we use the personal satisfaction and the emission of local signals to integrate a learning module in this model. The aim is to improve cooperative behaviors.

3. LEARNING TO COOPERATE

3.1 Principle of the approach

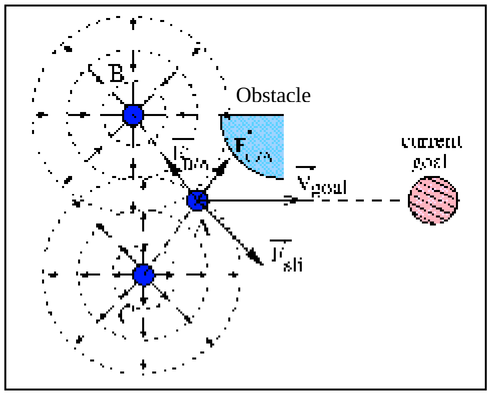

Figure 2 – An agent A perceiving two signals (an attraction from B and a repulsion from C), an obstacle and its current goal. If one signal is stronger than the personal satisfaction of A, the agent will change its goal to reply by computing the altruism vector (eq. 2).

We now extend the presented model to include learning, an ability that allows the agents to acquire new and adapt old behaviors for individual and group benefit.

The reinforcement learning approach [10][11] is well adapted to manage learning behavior selection. The simplicity of the algorithm and its possible use in dynamic non-deterministic environments makes it popular in many works [2][5][12].

In this method, the agent learns from external feedback received from the environment. The feedback is interpreted as a positive or negative scalar reinforcement. The goal of the learning system is to improve the action selection process by maximizing the externally given rewards.

The main issue in this approach is the evaluation of the rewards. In real systems, the agents must perceive themselves the success or the failure of their actions. In simulated grid worlds, reward values can be easily computed [13][14]. This problem remains difficult when it concerns the evaluation of agents interactions but it can be handled with communication.

In our approach, to make agents learn to cooperate, the reinforcement reward value is calculated using the neighbors personal satisfaction. Then, in the learning model, agents continuously emit their personal satisfaction level P : I(t) = P(t).

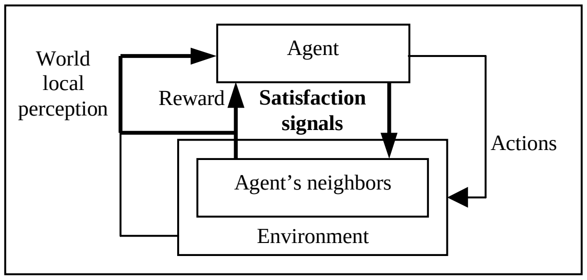

Figure 3 – Representation of the reinforcement learning model

Figure 3 abstracts the principle of the model. The following example gives an illustration: If an agent A acts on the environment and helps another agent B, the satisfaction level of B rises. By receiving B signals, agent A knows that its behavior is a cooperative one. By learning this behavior, agent A may reproduce it when the same situation is recognized.

Our learning model is an "on policy" method. This allows the agent to explore its action space and also to use the learned knowledge. The advantage of using an "on policy" method is that the agent continuously updates its knowledge following the environment evolution.

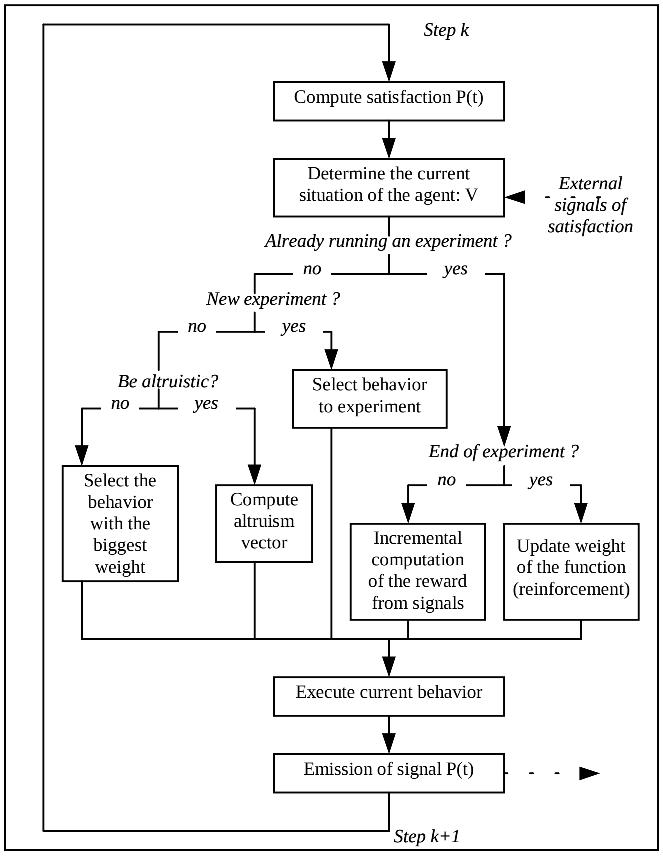
Step k
Compute satisfaction P(t)
Determine the current situation of the agent: V
External signals of satisfaction
Already running an experiment ?
no
yes
New experiment ?
no
yes
Be altruistic?
no
yes
Select behavior to experiment
End of experiment ?
no
yes
Select the behavior with the biggest weight
Compute altruism vector
Incremental computation of the reward from signals
Update weight of the function (reinforcement)
Execute current behavior
Emission of signal P(t)
Step k+1

Figure 4 – Agent's behavior algorithm

3.2 The formal model

Figure 4 presents the algorithm of the behavioral cycle of each agent. We detail below the more important stages of this cycle:

Agent's interactive state representation

The signals of personal satisfaction exchanged by the agents allow us to define a compact representation of their interactions.

We consider heterogeneous systems that are composed of n different types of agents. All emitted signals contain the sender type. Every agent holds a vector V of n values which represents the perceived signals. Each element v_i can have one of the three following values:

- 0 if no signal is received from any agent of type i
- 1 if the sign of the signal is positive
- 2 if the sign of the signal is negative

When several signals of the same type are received, only one is considered according to the following rule:

If several signals are received at the same time, the agent selects the more negative signal in priority over the more positive.

This rule ensures that conflicting situations (dissatisfied agents) are treated in priority. In situated multi-agent systems it is necessary to avoid any deadlock/conflict before providing other proprieties [4].

The current state of the agent is defined by the vector V and by other perceptions of the world. This representation of the interactions defines a finite set of conditions to trigger the behaviors (cf. application in section 4).

Triggering conditions of a learning phase

When an agent ends a behavior and perceives any signal, it decides following a probability (the exploration factor, 0.5 in simulations) to perform a learning phase (an experiment). In this case, the agent has to choose one of its behaviors.

The probability to select a function is inversely proportional to its number of trials. This weighting ensures the exploration of all the possible behaviors.

Reinforcement computing

During the experiment, an agent computes the average value of the satisfaction signals it receives. It applies the selection rule seen above in order to reinforce only the functions that satisfy all its neighbors.

This average value, noted $\overline{SatI}$ is directly used in the reinforcement computation. However, an average value may not be significant of the improvement of a situation. To avoid this problem, the reinforcement value is computed as the difference between the average satisfaction and the first received signal value (noted $SatI_0$). This difference is computed by:

$$\Delta = \frac{1}{2.P_{\max}}.\left(\overline{SatI} - SatI_0\right),\ \Delta \in [-1,1]$$

The r reward of the experiment depends on Δ if |Δ| is important (close to 1). Otherwise it depends essentially on the average value. Then the reward is computed as:

$$r' = |\Delta|.\Delta + (1-|\Delta|).\frac{\overline{SatI}}{P_{\max}}$$

This value must be normalized to weight interval definition, i.e. in [0,1]: $r = 0.5\ (r'+1)$. Thus $r \in [0,1]$, the weight of the experimented function is reinforced as follows:

$$W_i = b.W_i + (1-b).r$$

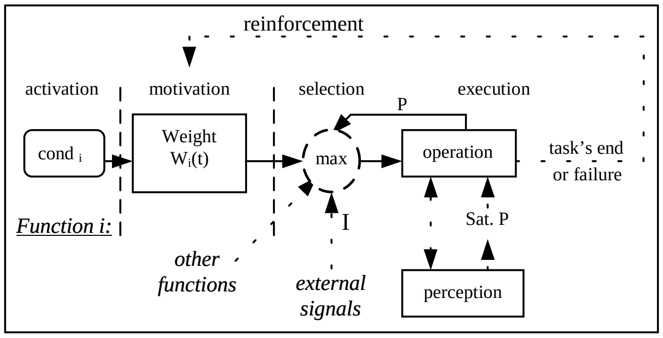

Figure 5 – Agent's behaviors representation, selection and reinforcement

The figure 5 abstracts the architecture and the reinforcement principle.

4. APPLICATION

We evaluated our learning model following classical collective tasks. More precisely, we studied variants of the foraging problem. This problem [15] can be presented as a base from which mobile robots must explore the world. These robots don't know the environment and only have a local perception of it. They must find some specific elements, take them and get them back to the base.

4.1 Pusher and cutter robots

We have defined a heterogeneous version of the foraging problem that introduces:

- The elements to be discovered are rectangular plates of any size (which represent for example toxic waste).
- There are two kinds of robots: the first one, the *pushers*, can take plates and move them to the base, the other kind, the *cutters*, can divide a plate into two new ones.
- The environment contains some obstacles (walls).

The *pusher robots* have an arm that allows them to catch a plate and apply a pushing force. So their capacity to move an object alone is limited. For big plates, the push power of several robots can be cumulated to allow their movement (this is the so-called box pushing application [16]).

The *cutter robots* have a tool to cut plates. They also have an arm that allows them to hold the plate and easily cut it. The division of a plate in two smaller ones simplifies the *pushers* job. Finally each agent has the six following behaviors:

Table 1 – Agent's behavior set

0: Random walk (plate searching)	3: Go to a detected plate
1: Push a plate	4: Hold a plate to stabilize it
2: Cut a plate (only for cutter robots)	5: Altruism reaction

4.2 Simulation rules

Robots evolve in a finite size world which has a rectangular shape and which is non-cyclic (world limits are considered as walls). The environment contains walls that robots can't move or cross.

Robots must move plates to a depot zone which is a disc centered on the base. If a plate stands in this zone, the base automatically consumes it (the plate disappears). To locate the depot zone, the base emits a signal continuously allowing robots to perceive its direction. A simulation is complete when all plates have been moved to the depot zone.

Personal satisfaction computation

The personal satisfaction of the agents is defined by an integer value in the interval [-127,127].

Computation of the satisfaction variation (eq. 1) for each behavior:

- search moving (for the two types of robots):

$$v = \begin{cases} 2.(0.5+\cos(a)) & \textit{if agent moves} \quad (m,n) \\ -3 & \textit{if agent cannot move} \quad (f) \end{cases}$$

  α is the angle difference between the performed moving direction and the goal direction.
- force application on a plate:

$$v = \begin{cases} 20 & \textit{if the box moves} \quad (m,n) \\ -20 & \textit{otherwise} \quad (f) \end{cases}$$

- cut action on a plate (only for *cutter robots*):

$$v = \begin{cases} 10.s & \textit{if cutting is possible} \quad (m,n) \\ -20 & \textit{otherwise} \quad (f) \end{cases}$$

  where *s* is the cutting speed defined by the interval]0,2]

Interactions

A plate can be cut by a single *cutter robot* at once. During a cutting phase :

- If another *cutter* tries to start a cut phase, this action fails and the first *cutter* is slowed down (*s*=0.5).
- If another *pusher* tries to move the plate, the cutting process stops (then both agents satisfaction drops).
- If another robot holds the plate it prevents any movement and speeds up the cut (*s*=2). Then the *cutter* satisfaction increases.

Current agent's situation representation

Now we apply to this system the agent's situation representation detailed in section 3.2. It is defined by two kinds of perceptions: perceived signal sign and plate perception (no plate, plate seen, plate handled) as shown in table 2.

Table 2 – Situation representation

Code	I1: cutter signal sign	I2: pusher signal sign	S3: plate percept.
0	No signal	No signal	No plate
1	Positive +	Positive +	Seen
2	Negative -	Negative -	Handled

These three variables allow us to represent in only 27 (3^3) different cases the interactive situations of each agent.

This representation leads us to define a behavioral matrix. It contains the trigger weight for each situation-function couple (as used by M. Matarić in [2]).

Matrixes are initialized in order to reproduce the behavior of the initial model without learning. The altruism function has a weight of 0.6, the random walk function is initialized to 0.1 and for every other case, an average weight of 0.5 is set.

4.3 Simulations and results

To experiment our learning method, we programmed a simulator of situated heterogeneous agents in dynamical environments (non-deterministic). This allows us to obtain realistic simulations of interactions between agents and their environment (see figure 6). The simulator is programmed in Java language on the MadKit platform [17].

Simulation of agents without learning module

The satisfaction model without learning allows cooperative interactions. We have observed the following cooperative behaviors:

- Conflict avoiding: if an agent A is hindered by another agent B, it emits a repulsive signal. By applying the altruistic reaction, agent B runs away.
- Cooperative help: if an agent doesn't succeed in pushing a plate (because of its weight), it emits an attractive signal to recruit other agents.
- Force summation: Several robots can cumulate their force in order to move a plate which is too heavy for a single robot (see figure 6.c).

But simulations on various environments show us that all conflicts are not well handled. For example: when a robot tries to push a plate on which a *cutter* works, it stops the cutting process. Holding the plate to ensure a fast cut would be a better behavior. The goal of the learning module is to discover these new cooperative behaviors or to optimize existing ones to handle conflicting cases.

Taking advantage of the Learning module

Simulations were performed on the environment shown in Figure 6, which contains a big plate to be cut and moved. This small environment has been designed to cause many agent interactions.

Behavioral matrixes evolved during several simulation runs. After few successive simulations (about 5), we observed some behaviors that were not programmed in the initial behavioral matrixes. Obtained values became stabilized after *only thirty to forty experiments* of the same function. We present here the more significant results :

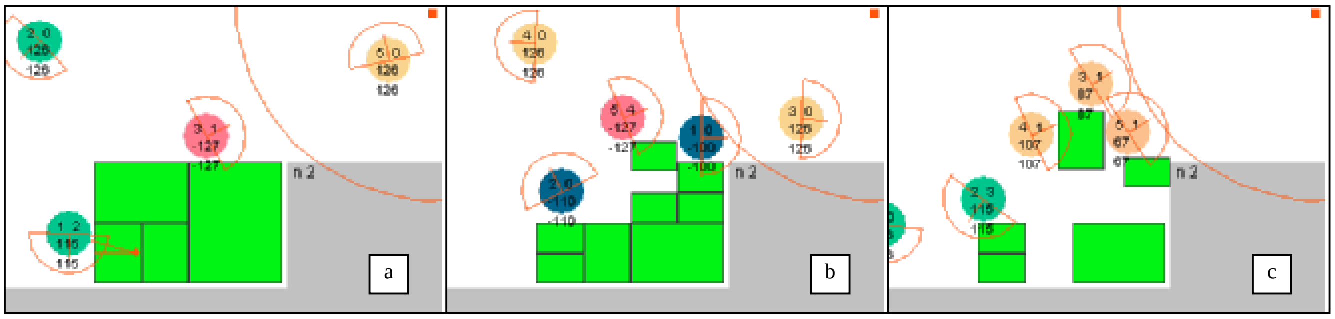

Figure 6 – Simulation evolution snapshots (the personal satisfaction signal is written under each agent, the half-disc represents the agent perception)

- *Pusher robots* reinforced their pushing function for situations where another *pusher* was perceived.
- Both robot types decreased the altruistic function weight when *pusher* neighbors were dissatisfied, because running away does not help the other *pushers*.
- *Cutter robots* decreased Hold and Cut functions when these behaviors were not compatible with their neighbors.
- *Cutter robots* reinforced the Hold function to stabilize the plate when another *cutter* was working. This behavior accelerated the cutting process.
- *Cutter robots* discovered a new behavior not initially programmed. When they were close to *pusher robots*, *cutters robots* also learned to push the plate. This behavior helped the *pusher* to move heavy plates (see table 3).

Table 3 – *Cutter* discovered the Push function

Agent n1 in situation 022	Before	After	Δ
Random walk	0,10	0,11	+ 10%
Push a plate	0	**0,12**	**New**
Cut a plate	0,50	0,09	- 82%
Go to detected plate	/	/	/
Hold and stabilize plate	0,50	0,15	- 70%
Altruism reaction	0,60	0,10	- 83%

Consider the situation 022 (table 3) where a *cutter robot* holds a plate and perceives a *pusher* neighbor that is not satisfied. The push function with an initial weight of 0 has been experimented to reach a weight of 0.12 in only 35 tries. That makes this function a predominant one for this situation. This behavior can be translated by: "If I hold a plate and if I perceive a signal of a dissatisfied *pusher*, I push the plate to satisfy it".

More simulation results and their analysis can be viewed on the web site http://www.lirmm.fr/~chapelle/works.

5. CONCLUSION

This work focused on learning cooperative behaviors in situated multi-agent systems.

We presented an architecture of cooperative agents based on a personal satisfaction measure and on local signal diffusions. Then we introduced a reinforcement learning module to improve this architecture. Agent states were defined by an original compact representation of its perceived signals. Moreover, behavior weights of the agents were reinforced by monitoring neighbors satisfactions.

This model has been evaluated by simulation of a heterogeneous variant of the foraging problem. Analysis of the behavioral matrixes of the agents has shown the apparition of new cooperative behaviors and weight reduction of others. Thus many conflicting situations, not fully handled in the satisfaction-altruism reactive model, are now better managed.

Evaluation of the model is still pending. Many different environments are currently being tested to demonstrate its usefulness. Moreover, we think to improve the model by handling dependencies between tasks as proposed in [13][14].

However, these preliminary results seem very encouraging to us. As a consequence, we envisage to implement the presented learning module on real autonomous robots. Actually, the satisfaction-altruism architecture is already implemented and validated on real robots [4]. Which therefore should allow us to integrate the proposed learning module easily.

6. REFERENCES

[1] R. Arkin, *Motor schema based mobile robot navigation*, International journal of robotics research, 92-112, 1989.

[2] M.J. Matarić, *Interaction and intelligent behavior*, MIT EECS PhD Thesis, MIT AI Lab, 1994.

[3] L. Parker, *ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots*, Proc. of the 1994 International Conference on Intelligent Robots and Systems, 776-783, 1994.

[4] P. Lucidarme, O. Simonin and A. Liegeois, *Implementation and Evaluation of a Satisfaction-Altruism Based Architecture for Multi-Robot Systems*, in International Conference on Robotics and Automation, to appear, 2002.

[5] T. Shibata, K. Ohkawa, K. Tanie, *Spontaneous behavior of robots for cooperation – emotionally intelligent robot system –*, Proceedings of the IEEE Int. Conf. on Robotics and Automation, 2426-2431, 1996.

[6] A. Drogoul, J. Ferber, *From tom thumb to the dockers: some experiments with foraging robots*, in 2nd Int. Conference On Simulation of Adaptative behavior, 451-459, 1992.

[7] O. Simonin and J. Ferber, *Modeling self satisfaction and altruism to handle action selection and reactive cooperation*, 6th International Conference on the Simulation of Adaptive Behavior, proceedings Supplement, 314-323, 2000.

[8] O. Simonin, A. Liegeois and P. Rongier, *An architecture for Reactive Cooperation of Mobile Distributed Robots*, in Distributed Autonomous Robotic Systems 4, 35-44, 2000.

[9] M.J. Matarić, *Learning to Behave Socially*, From animals to animats, 453-462, April 1994.

[10] R.S. Sutton, A.G. Barto, *Reinforcement Learning: an Introduction*, MIT Press, Cambridge, MA, 1998.

[11] L.P. Kaebling, M.L. Littman, A.W. Moore, *Reinforcement Learning : A survey*, Journal of Artificial Intelligence Research 4, 237-285, 1996.

[12] L.E. Parker, *L-ALLIANCE: Task-Oriented Multi-Robot Learning in Behavior-Based Systems*, Advanced Robotics, Special Issue on Selected Papers IROS '96, 305-322, 1997.

[13] A. Dutech, O. Buffet, F. Charpillet, *Multi-Agent Systems by Incremental Gradient Learning*. Int. Joint Conference on AI, 2001

[14] C. Boutilier, *Sequential Optimality and Coordination in Multiagent Systems*. In Proceedings of the 16th Int. Joint Conference on AI, 1999.

[15] L. Steels, *Cooperation between distributed agents through self-organization*. In Decentralized AI-Proceedings of First European Workshop on Modeling Autonomous Agents in a Multi-Agent World, 175-196. Elsevier Science B.V., Amsterdam. 1989.

[16] C.R. Kube, H. Zhang, *Collective robotic intelligence*, in 2nd International Conference on the Simulation of Adaptive Behavior (SAB), Honolulu, HI. MIT Press, Cambridge, 1992.

[17] O. Gutknecht, J. Ferber, *The MadKit agent platform architecture*, 1st Workshop on Infrastructure for Scalable Multi-Agent Systems , 2000.