A Linearly Quasi-Anticipatory Autonomous Agent Architecture: Some preliminary experiments

Paul Davidsson

Dept. of Computer Science, Lund University, Box 118, S–221 00 Lund, Sweden

Abstract. This report presents some initial results from simulations of a linear quasi-anticipatory autonomous agent architecture (ALQAAA), which correspond to a special case of a previously suggested general architecture of anticipatory agents. It integrates low-level reaction with high-level deliberation by embedding an ordinary reactive system based on situation-action rules, called the Reactor, in an anticipatory agent forming a layered hybrid architecture. By treating all agents in the domain (itself included) as reactive agents, this approach drastically reduces the amount of search needed while at the same time requiring only a small amount of heuristic domain knowledge. Instead it relies on a linear anticipation mechanism, carried out by the Anticipator, to achieve complex behaviours. The Anticipator uses a world model (in which the agent is represented only by the Reactor) to make a sequence of one-step predictions. After each step it checks whether the simulated Reactor has reached an undesired state. If this is the case it will modify the actual Reactor in order to avoid this state in the future. Results from both single- and multi-agent simulations indicate that the behaviour of ALQAAA agents is superior to that of the corresponding reactive agents. Some promising results on cooperation and coordination of teams of agents are also presented. In particular, the linear anticipation mechanism is successfully used for conflict detection.

1 Introduction

In this report we will present some initial results from experiments on a linearly quasi-anticipatory autonomous agent architecture. This architecture corresponds to a special case of the general architecture of anticipatory agents suggested by Astor, Davidsson, and Ekdahl [1, 4].

1.1 Hybrid Agent Architectures

In the last couple of years it has become widely acknowledged that an intelligent autonomous agent must have the capability of both high-level deliberation and low-level reaction. This insight can be seen as a synthesis of two earlier rival lines of thought: the first argued that agent cognition should be based on sophisticated deliberation, typically planning, and the second that it should be based only on primitive reactive behaviour specified for instance by situation-action rules. As it turns out, the weaknesses of reactive approach correspond closely to the strengths of deliberative approach and vice versa, i.e., reactive agents are fast but "dumb" (and do not need an explicit world model) whereas deliberative agents are "smart" but slow (and need a detailed world model).

Moreover, a combined, or *hybrid*, approach seems to model human functioning closer than the purely reactive approach, which resembles that of more primitive animals.

As Hanks and Firby [8] point out, two categories of hybrid agent architectures can be distinguished: *uniform* architectures employ a single representation and control scheme for both reaction and deliberation, whereas *layered* architectures use different representations and algorithms implemented in separate layers to perform these functions. Most uniform architectures, for example PRS [10], do not make any specific commitments on how reaction and deliberation should be interleaved. In addition, Ferguson [5] argues that "There are a number of other reasons for advocating a layered control approach, including increased behavioural robustness and operational concurrency, as well as improved program comprehensibility and system testability and analysability." (p.48)

However, as pointed out by Wooldridge and Jennings [17] there is a weakness in the existing suggestions for hybrid architectures. They argue that:

> One potential difficulty with such architectures, however, is that they tend to be *ad hoc* in that while their structures are well-motivated from a design point of view, it is not clear that they are motivated by any deep theory. (p.26)

In order to improve this situation, a layered hybrid approach based on the theory of anticipatory systems will be presented in the next section.

1.2 Anticipatory Agents

According to Rosen [15], an anticipatory system is " ... a system containing a predictive model of itself and/or of its environment, which allows it to change state at an instant in accord with the model's predictions pertaining to a latter instant." (p.339) Thus, such a system uses the knowledge concerning future states to decide which actions to take in the present.

Let us describe a simple class of anticipatory systems suggested by Rosen [14]. It contains an ordinary causal (i.e., non-anticipatory) dynamic system, S. With S he associates another dynamical system, M, which is a model of S. It is required that the sequence of states of M are parameterized by a time variable that goes faster than real time. (That is, if M and S are started out at some time t_0, then after an arbitrary time interval Δt, M's sequence of states will have proceeded $t_0 + \Delta t$.) In this way, the behaviour of M predicts the behaviour of S: by looking at the state of M at time t, we get information about the state that S will be in at some later time than t. In addition, M is equipped with a set E of effectors which allows it to operate either on S itself, or on the environmental inputs to S, in such a way as to change the dynamical properties of S. If S is modified the effectors must also update M. This class of anticipatory systems is illustrated in Fig. 1. Rosen argues that this would be an anticipatory system in the strict sense if M were a perfect model of S (and if the environment were constant or periodic). However, as M in general is not a perfect model of S, he calls the behaviour of such a system *quasi-anticipatory*.

Then, how should these predictions be used to modify the properties of S? Rosen [14] argues that this could be done in many ways, but suggests that the following is the simplest:

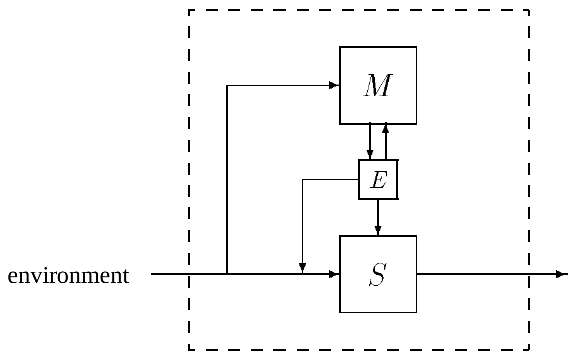

Fig. 1. The basic architecture of a class of anticipatory systems.

> Let us imagine the state space of S (and hence of M) to be partitioned into regions corresponding to “desirable” and “undesirable” states. As long as the trajectory in M remains in a “desirable” region, no action is taken by M through the effectors E. As soon as the M-trajectory moves into an “undesirable” region (and hence, by inference, we may expect the S-trajectory to move into the corresponding region at some later time, calculable from a knowledge of how the M- and S-trajectories are parameterized) the effector system is activated to change the dynamics of S in such a way as to keep the S-trajectory out of the “undesirable” region. (p.248)

However, to transfer the concept of anticipatory systems into an agent framework there are some additions and changes that we have found necessary. First, we need a meta-level component that “runs” and “monitors” the model, and evaluates the predictions made to decide how to change S (or the input to S, we will in the following regard also such changes as changes to S). Thus, we also include the effectors, E, in this meta-level component that we here will call the *Anticipator*. Second, in order to predict future environmental inputs to S we need to extend the model M to include also the environment. This inclusion is in line with later work of Rosen (cf. [15]). In sum, an *anticipatory agent* consists mainly of three entities: an object system (S), a meta-level component (Anticipator), and a world model (M). The object system is an ordinary (i.e., non-anticipatory) dynamic system. M is a description of the environment *including* S, but excluding the Anticipator. The Anticipator should be able to make predictions using M and to use these predictions to change the dynamic properties of S. Although the different parts of an anticipatory agent certainly are causal systems, the agent taken as a whole will nevertheless behave in an anticipatory fashion.

When implementing an anticipatory agent, what should the three different components correspond to, and what demands should be made upon these components? To begin with, it seems natural that S should correspond to some kind of reactive system similar to the ones mentioned above. We will therefore refer to this component as the *Reac-*

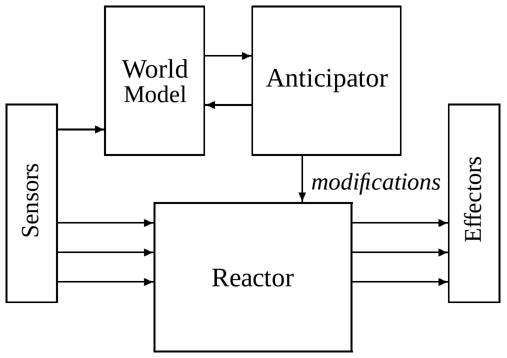

Fig. 2. The basic architecture of an anticipatory agent.

tor. It should be a fast system in the sense that it should be able to handle routine tasks instinctively and, moreover, it should have an architecture that is both easy to model and to change. The Anticipator would then correspond to a more deliberative meta-level component that is able to "run" the world model faster than real time. When doing this it must be able to reason about the current situation compared to the predicted situations and its goals in order to decide whether (and how) to change the Reactor.

The resulting architecture is illustrated in Fig. 2. To summarize, the sensors receive input from the environment. This data is then used in two different ways: (1) to update the World Model and (2) to serve as stimuli for the Reactor. The Reactor reacts to these stimuli and provides a response that is forwarded to the effectors, which then carry out the desired action(s). Moreover, the Anticipator uses the World Model to make predictions and on the basis of these predictions the Anticipator decides if, and what, changes of the dynamical properties of the Reactor are necessary. Every time the Reactor is modified, the World Model should, of course, be updated accordingly. Thus, the working of an anticipatory agent can be viewed as two concurrent processes, one reactive at the object-level and one more deliberative at the meta-level.

2 A Simple Class of Anticipatory Agents

In this section some initial results from experiments with A Linearly Quasi-Anticipatory Agent Architecture (ALQAAA) will be presented. ALQAAA corresponds to a special case of the architecture described in the last section. In particular, it follows Rosen's suggestion of the simplest way of deciding when to change the Reactor, i.e., by dividing the state space into desired and undesired regions.

Some simple ALQAAA-agents and a testbed has been implemented.[1] The problem

[1] They have been implemented in the object-oriented language Simula on Sun SparcStation running Solaris 2.3. Local class packages were used to achieve concurrency (Simlib IOProcesses) and graphical interface to X (WindowPackage).

domain has deliberately been made as simple as possible in order to make the principles of anticipatory behaviour as explicit as possible.

2.1 Agent Implementation

The Reactor and the Anticipator are run (asynchronously) as two separate processes. The Reactor process is given a high priority whereas the Anticipator is a low priority process that runs whenever the Reactor is "waiting" (e.g., for an action to be performed). Since the Reactor is able to preempt the Anticipator at any time, reactivity is always guaranteed. Thus, the Anticipator has to be a kind of *anytime algorithm* [2], or rather anytime process, in that it should always be able to return a result when it is interrupted.[2] The appropriateness of using anytime algorithms in autonomous agent contexts where real-time requirements are common has been pointed out by, for example, Zilberstein and Russell [18] and Bresina and Drummond [3].

The Reactor carries out a never ending cycle of: perception of the environment (i.e., the situation), action selection by situation-action (stimuli-response) rules, and performance of action. Rather than having explicit goals, the Reactor's goals are implicitly represented in its collection of situation-action rules. The basic algorithm of the Reactor is given below:

```
procedure REACTOR;
while true do
    Percepts ← Percieve;
    Action ← SelectAction(Percepts);
    Perform(Action);
```

The Anticipator, on the other hand, carries out a never ending cycle of anticipation sequences. Each such sequence begins with making a copy of the World Model, which as mentioned earlier is a description of the environment containing the agent as a physical entity in the environment, and a copy of the agent's current set of reaction rules. These are then used to make a sequence of one-step predictions. After each prediction step, it is checked whether the simulated agent has reached an undesired state, or whether it has achieved the goal. If it has reached an undesired state, the Reactor will be manipulated in order to avoid reaching this state. Thus, this functioning corresponds to that of the simplest kind of anticipatory system suggested by Rosen. The basic algorithm of the Anticipator is as follows:

[2] According to Dean and Boddy [2], the main characteristics of anytime algorithms are that "... (i) they lend themselves to preemptive scheduling techniques (i.e., they can be suspended and resumed with negligible overhead), (ii) they can be terminated at any time and will return some answer, and (iii) the answers returned improve in some well-behaved manner as a function of time." (p.52)

```
procedure ANTICIPATOR;
while true do
    WorldModelCopy ← WorldModel;
    ReactorCopy ← Reactor;
    UndesiredState ← false;
    while not UndesiredState and not GoalAchieved(WorldModelCopy) do
        Percepts ← WorldModelCopy.Percieve;
        Action ← ReactorCopy.SelectAction(Percepts);
        WorldModelCopy.Perform(Action);
        UndesiredState ← Evaluate(WorldModelCopy);
    if UndesiredState then
        Manipulate(Reactor);
```

Note that since the behaviour of the Reactor in each situation is determined by situation-action rules, the Anticipator always “knows” which action the Reactor would have performed. Also the environment (including all other agents) is treated as being purely reactive. Thus, since everything is governed by situation-action rules, the anticipation mechanism requires no search, or in other words, the anticipation is *linear*. It should also be noted that the goal of the agent is not limited to pick up only one target (i.e., a singular goal). In a multi-goal scenario, some of the changes (manipulations) of the Reactor should only hold for a limited interval of time (e.g., until the current goal has been achieved). Otherwise, there is a danger that these changes might prevent the agent to achieve other goals.

In more formal terms a linearly quasi-anticipatory agent can be specified as a tuple:

$$\langle \mathcal{R}, \mathcal{W}, \mathcal{U}, \mathcal{M} \rangle$$

where

$\mathcal{R}$ is the set of situation-action rules defining the Reactor.
$\mathcal{W}$ is the description of the environment (the world model).
$\mathcal{U}$ is the set of undesired states.
$\mathcal{M}$ is the set of rules describing how to modify $\mathcal{R}$.

The Anticipator is defined by $\mathcal{U}$ and $\mathcal{M}$. For each element in $\mathcal{U}$ there should be a corresponding rule in $\mathcal{M}$, which should be applied when an undesired state of this kind is anticipated. Thus, we need in fact also a function, $f : U \rightarrow M$, that determines which rule for modifying the Reactor that should be applied given a particular type of undesired state. However, as this function typically is obvious, it will not be described explicitly. Moreover, in all simulations described below, $\mathcal{W}$ will consist of the positions of all obstacles, targets, and agents present in the environment.

Using these terms, the function Evaluate can be described as checking whether the current anticipated state belongs to $\mathcal{U}$, and Manipulate as first applying f on the anticipated undesired state and then using the resulting rule from $\mathcal{M}$ to modify $\mathcal{R}$.

2.2 The Testbed

The agent's environment is a two-dimensional grid (10×10) in which a number of unit-sized square obstacles forms a maze. In addition to these static objects, there are two kinds of dynamic objects: agents, which can move about in the maze, and targets, which can be removed by an agent.

The goal of an agent is to pick up the target(s). To be able to pick up a target, the agent must be in the same position as the target. The agent is able to move in four directions (north, south, east, and west), unless there is an obstacle that blocks the way. The agent is always able to perceive the target (i.e., the angle to the target), and whether there are any obstacles immediately north, south, east, or west of the agent. (You can think of the obstacles as being made of glass: it is possible to see the targets through the obstacles but not the obstacles themselves, which only can be perceived by tactile sensors.)

Almost identical environments have been used in other experiments, for instance by Sutton [16]. It is also similar to the Tileworld [13] (if we interpret the targets as holes) except for that: (i) following Kinny and Georgeff [11] our testbed has no tiles as they only make the simulations unnecessarily complex and (ii) some randomness have been excluded (e.g., the pattern-less appearance and disappearance of holes) since it makes much of the deliberation pointless (i.e., prediction becomes impossible, cf. Hanks [9]). In Section 2.4 the environment is generalized into a multi-agent scenario (as suggested by, for instance, Zlotkin and Rosenschein [19]). The advantages and disadvantages of this kind of testbeds have been discussed at length by Hanks, Pollack and Cohen [9].

2.3 Single Agent Simulations

In this section some experiments on single agents will be presented. We will compare the performance of reactive agents with that of ALQAAA agents containing such agents as their Reactor.

Let us consider a simple reactive agent that has only one simple situation-action rule: *move to the free position closest to the target*. In other words, it first tries to move to the adjacent position that reduces the distance to the target the most. If there is a block in this position, it tries the direction of the remaining three that reduces the distance if there are one, else it tries the position that increases the distance the least and so on. A reactive agent of this kind will behave rather well in many not too complicated mazes. However, as illustrated in Fig. 3 (a) there are also very simple mazes in which it behaves poorly. Starting out in the position marked A, it will move two positions towards the target (T) and then "loop" between two positions as marked in the figure.

Let us now consider an ALQAAA agent that has this reactive agent as its Reactor. That is, $\mathcal{R}$ = {*move to the free position closest to the target*}. We define the undesired states as those in which the agent is trapped in a loop, and if such a state is detected by the Anticipator, the Reactor is manipulated in such a way that it will not enter this state from now on. Thus, we have that: $\mathcal{U}$ = {*being in a loop*} and $\mathcal{M}$ = {*avoid the position in the loop closest to the target*}.[3] An ALQAAA agent of this kind will then behave as showed in Fig. 3 (b). The Anticipator will by anticipation detect the loop before entering it and

[3] Which position in the loop to avoid can, of course, be selected according to other principles.

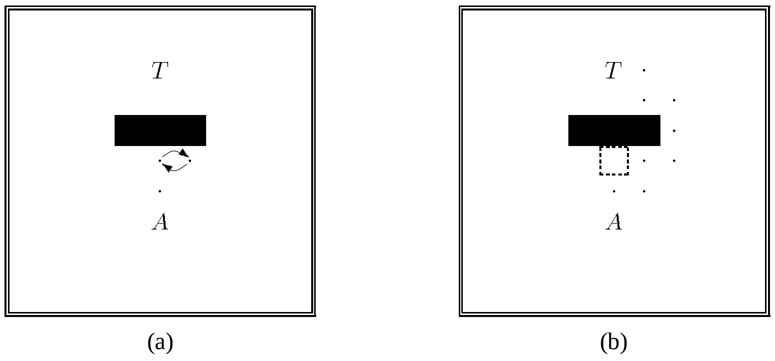

Fig. 3. The behaviour of (a) the Reactive agent and (b) the ALQAAA agent. A indicates the agent's present position and T the position of the target.

make the Reactor avoid the position closest to the target (marked by a dashed box). This is sufficient for the Reactor to find its way to the target.

We have compared the performance of a reactive and an ALQAAA agent of the kinds described above in a series of experiments. There was one target in the environment and the number of obstacles varied between 0 and 35. In each experiment the positions of the agent, the target and the obstacles were randomly chosen. In order to avoid too many trivial scenarios there was also a requirement that the distance between the agent and the target should be at least five unit-lengths. Moreover, only scenarios in which it is possible for the agent to reach the target were selected. From the result in Fig. 4, we see that the more complex the environment gets, the more useful is the anticipatory behaviour. If there are no obstacles at all, even the Reactor will, of course, always find the target.

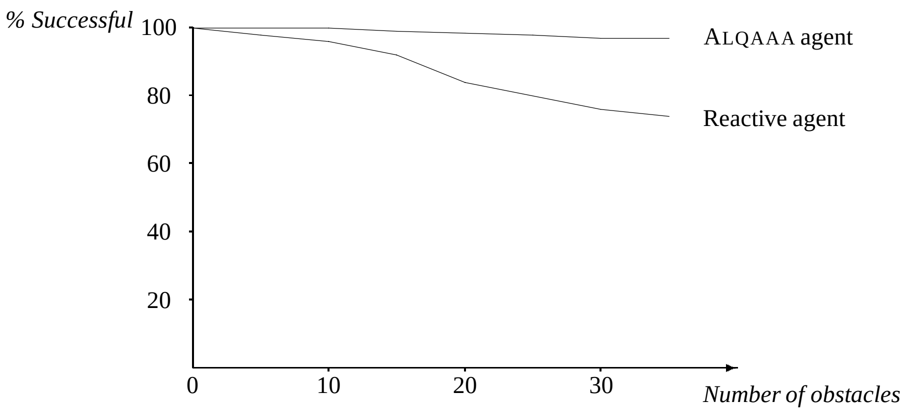

Fig. 4. Comparison between Reactive and ALQAAA agents. (200 runs of each multiple of five)

This ALQAAA agent is able to reach the target (when this is possible) in almost all kinds of mazes. However, as Fig. 4 shows, there are some in which it will not succeed and they are typically of the kind depicted Fig. 5 (a). The reactive agent will in this case

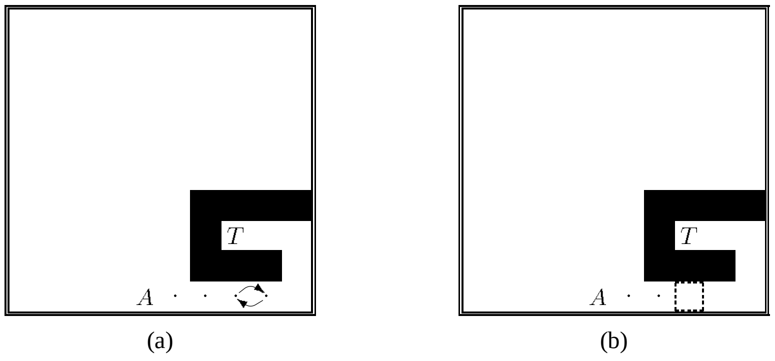

Fig. 5. The behaviour of (a) Reactive agent and (b) ALQAAA agent.

be trapped in a loop whereas the ALQAAA agent will detect this loop beforehand and block the position as shown in Fig. 5 (b). This implies that the only possible way to the target is blocked and the agent will never reach the target.

The problem with this Anticipator is that it is too eager to block a position. The reason for this is that the Reactor is inclined to "turn around" as soon as it is not decreasing the distance to the target. If we just augment the Reactor's situation-action rule with the condition that it should only change its direction 180° when there are no other alternatives (i.e., if there are obstacles in the three other directions), we will get a reactive agent that solves this problem. If we take this Reactor and the same Anticipator as used in the last example, we get an ALQAAA agent which seems always to reach the target (if the Anticipator is given enough time to anticipate, that is). This Reactor (i.e., $\mathcal{R} = \{$*move to the free position closest to the target, but do not turn around if not forced to*$\}$) will be used in all the experiments that follows.

2.4 Multi-Agent Simulations

In this section some experiments in multi-agent domains are presented. We will point out the advantages of being anticipatory, both when competing and when cooperating with other agents. While the experiments have been carried out with two competing or cooperating agents in order to make things as clear as possible, it is trivial to extend the experiments to permit a larger number of agents.

Competing Agents The main idea here is that an ALQAAA agent should use its model of other agents in order to detect future situations in which other agents interfere with the agent's own intentions (i.e., goals). When such a situation is detected, the Anticipator should manipulate the Reactor in order to minimize the probability that this situation will ever occur.

We will evaluate this approach in the same testbed as above but with two agents and more than one target in the environment. Agent A should be regarded as "our" agent, whereas agent B represents the agent with which it competes. The goal of both the agents is to pick up as many targets as possible. In addition to the basic algorithm described above, the Anticipator needs a model of the agent B which it uses to predict B's actions in the same manner as it predicts its own (i.e., A's) actions. When the Anticipator realizes that B will reach a target before A, it notifies the Reactor that it should ignore this target. Thus, we have that: $\mathcal{U}$ = {*being in a loop, pursuing targets that presumably will be picked up by another agent*} and $\mathcal{M}$ = {*avoid the position in the loop closest to the target, avoid the target that presumably will be picked up by another agent*}.

However, let us start with two reactive agents of the kind described above. An environment containing three targets is described in Fig. 6 (a). If the agents start at the same

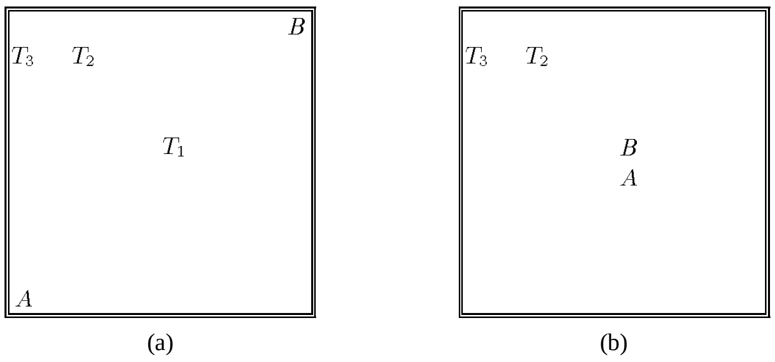

Fig. 6. The behaviour of two competing reactive agents: (a) the initial state (b) the situation after 8 time steps, agent B has picked up target T_1.

time the following will happen. Both agents perceive that T_1 is their closest target and they both head towards it. As B is somewhat closer to T_1 than A, it will reach it first and pick it up (see Fig. 6 (b)). B will then head for T_2 which now is the closest target. A will also head for T_2 following B. B then reaches T_2, picks it up, and heads for the last target T_3 with A still behind. Eventually B will pick up also T_3. Thus, B gets all the targets and A gets none.

If we, on the other hand, let A be an ALQAAA agent and start in the same position as above, it will soon detect that B will be the first to reach T_1. So, A will avoid this target and instead head towards T_3 (which is the next closest target to A). It will reach T_3 at the same time as B reaches T_1. When the agents have picked up their targets, there is only one target left (T_2). Since A is closest to T_2, it will reach it first. Thus, by anticipating the behaviour of both itself and the other agent, A will pick up two targets whereas B only picks up one.

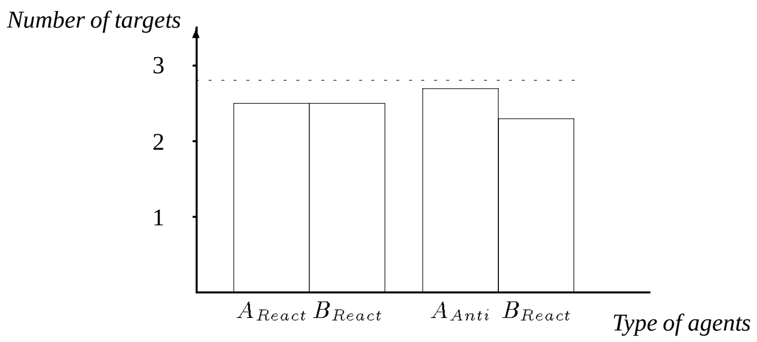

Fig. 7. Comparison between two sets of competing agents. To the left are both A and B reactive agents and to the right is A an ALQAAA agent and B a reactive agent. The vertical axis indicates the number of targets picked up by an agent (averages over 1000 runs). The optimal number of targets that A is able to pick up in this situation (i.e., given B's behaviour) is illustrated by the dashed line.

We have also some quantitative results on the superiority of ALQAAA agents when competing with reactive agents. In the experiments there were 30 obstacles, 5 targets and two agents. In the first session both the agents were reactive and in the second there was one ALQAAA agent competing with a reactive one. The results shown in Fig. 7 tell us that the performance indeed is improved (being almost optimal) when the agent behaves in an anticipatory fashion.

Cooperating Agents We shall now see how ALQAAA agents can be used for cooperating. The task for the two agents is to pick up all the targets in shortest possible time. It does not matter which agent that picks up a particular target.

To begin with, we apply the agents in the last example (i.e., A is an ALQAAA agent and B a reactive agent) to the situation described in Fig. 8 (a). As these agents are not cooperating, their global behaviour will (as one might expect) not be optimal. What will happen is that both agents initially head towards the same target (T_1). When agent A reaches T_1 we have the situation depicted in Fig. 8 (a). The other targets will then be approached in the same fashion, with one agent following the other. As a result, it will take these non-cooperating agents 15 time-steps to pick up all the targets.

Cooperating agents, on the other hand, should be able to make use of the fact that the ALQAAA agent knows that it will pick up the two closest targets. One way of doing this is to let agent A send a message to agent B (which still is a reactive agent) when it believes that it will pick up a particular target. This message contains the information that agent B should avoid this target. Thus, we add "*other agent pursuing target that presumably will be picked up by me*" to $\mathcal{U}$ and "*send message to other target that it should avoid the target*" to $\mathcal{M}$.

When this method is applied to the previous example, agent A will detect that it will pick up targets T_1 and T_2 and sends therefore messages to B that these should be avoided. B then directly heads towards T_2, which the only remaining target that A will

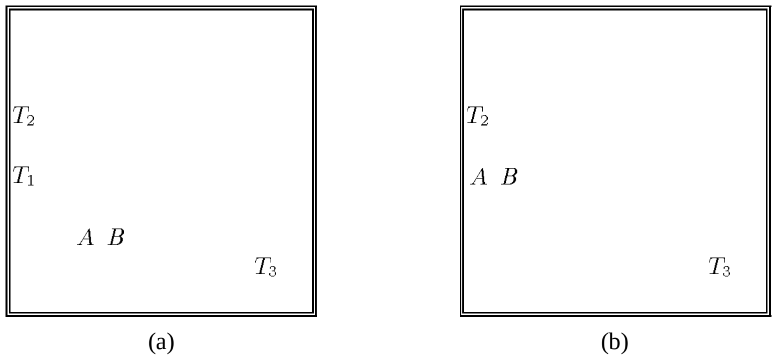

Fig. 8. The behaviour of two non-cooperating agents one ALQAAA (A) and one reactive (B). (a) the initial state (b) the situation after 4 time steps, agent A has picked up target T_1.

not reach before B. As a result, this system of cooperating agents will use only 6 time-steps to pick up all the targets.

The quantitative results (using 30 obstacles, 5 targets and two agents) are summarized in Fig. 9. We see that when the two agents are cooperating they come close to optimal behaviour.

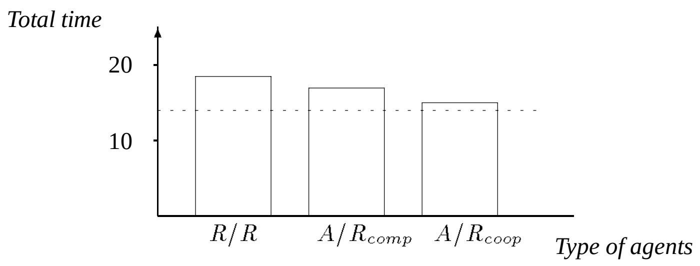

Fig. 9. Comparison between three sets of agents in terms of the total time it takes to collect all the five targets (averages over 1000 runs). R/R denotes two reactive agents, A/R_{comp}, one reactive and one ALQAAA agent competing, and A/R_{coop}, one reactive and one ALQAAA agent that are cooperating. The optimal time for two agents to collect all targets in this situation is illustrated by the dashed line.

In the scenario described here, it is only agent A that is an ALQAAA agent whereas B is an ordinary reactive agent. Even if we also let B be an ALQAAA agent with the same model of the world, we would not increase the performance. The reason for this is that both agents would have made the same predictions and therefore send messages to each other about things that both agents have concluded by themselves. However, in a more realistic setting where the agents do not have exactly the same information about the world, such an approach would probably be fruitful. In such a case things gets quite complicated if one ALQAAA agent simulates the anticipatory behaviour of another ALQAAA agent which in turn simulates first agent's anticipatory behaviour. The solution we suggest is to simulate only the reactive component of agents and let the agents communicate (e.g., broadcast) their modifications of their reactive component when they are performed to the other agents. In this way we are still able to make linear anticipations. This approach can be contrasted with the Recursive Modeling Method suggested by Gmytrasiewicz and Durfee [7] in which an agent modeling another agent includes that agent's models of other agents and so on, resulting in a recursive nesting of models.

As we have seen the behaviour of the ALQAAA agents implemented has not been optimal. However, we have deliberately chosen very simple Reactors and Anticipators for the purpose of illustrating how easily the performance can improved by embedding a Reactor in an ALQAAA agent. It should be clear that it is possible to develop more elaborate $\mathcal{R}, \mathcal{U}$, and $\mathcal{M}$ components that produce behaviour closer to the optimal.

3 Discussion

We have shown the viability of an approach for designing autonomous agents based on the concept of anticipatory systems called ALQAAA in a simple navigation task. Adaptation to the environment is performed by letting the Anticipator component of the agent manipulate the Reactor component according to anticipated future states.

Compared to traditional planning, anticipation as described here (there may be other ways to anticipate) is a more passive way of reasoning. The ALQAAA agent just tries to predict what will happen if nothing unexpected occurs, whereas a planning agent actively evaluates what will happen when a number of different actions are performed. The result is that planning agents will rely heavily on search, whereas ALQAAA agents will not. The main reason for this is that all agents in the environment (also the ALQAAA agent itself) are treated as being reactive.

In addition, it is interesting to note the small amount of heuristic domain knowledge that is given to the Reactor and the Anticipator (i.e., $\mathcal{R}, \mathcal{U}$, and $\mathcal{M}$). Thus, this approach drastically reduces the amount of search needed while at the same time requiring only a small amount of heuristic domain knowledge. Instead, it relies on a linear anticipation mechanism to achieve complex behaviours.

3.1 Related Work

The main task of the Anticipator is to avoid undesired states whereas the main task of the Reactor is to reach the desired state(s). In other words, the Anticipator's goals are goals of maintenance and prevention rather than of achievement. Compare this to Minsky's suppressor-agents, discussed within his Society of Mind framework [12], which waits until a "bad idea" is suggested and then prevents the execution of the corresponding action. However, there is a big difference, suppressor-agents are not predictive. The Anticipator takes actions beforehand so that the bad idea never will be suggested! Thus, an Anticipator can be regarded as predictive suppressor-agent.

In conformity with the Sequencer component in Gat's ATLANTIS architecture [6], the Anticipator can be viewed as being based on the notion of *cognizant failures* (i.e., a failure that can be detected by the agent itself). However, an Anticipator detects these failures in a simulated reality (i.e., a model of the world), whereas a Sequencer has to deal with real failures.

The notions of Reactor and Anticipator have some similarities with the Reactor and Projector components in the ERE architecture suggested by Bresina and Drummond [3]. In particular, the Reactor in ERE is able to produce reactive behaviour in the environment independently, but also takes advise from the Projector based on the Projector's explorations of possible futures. However, the Projector is more similar to a traditional planner in that it is based on search through a space of possible Reactor actions (a third component, the Reductor, is introduced to constrain this search), whereas the Anticipator simulates the behaviour of the Reactor in its environment linearly (i.e., without search). Moreover, the Anticipator's main task is to avoid undesired states, whereas the Projector in the ERE tries to achieve desired states.

There are also some similarities to the DYNA architecture [16] if we let the Reactor correspond to DYNA's Policy component and the Anticipator to its Evaluation function. In DYNA two kinds of rewards can be identified: *external* rewards, which are those that the Evaluation function gets from the environment (this kind of rewards is not required by an ALQAAA agent), and *internal* rewards, which are those that the Policy gets from the Evaluation function (these can be compared with the manipulations that the Anticipator performs on the Reactor). However, there are many disadvantages with the DYNA architecture compared to ALQAAA agents: (i) The planning process in DYNA requires search (in fact, random search) when it internally tests the outcome of different actions. Moreover, several trials are typically necessary whereas ALQAAA agents only need one. (ii) Even if the goal is changed only slightly (e.g., the target is moved one position), the learning in DYNA must start again from scratch. (iii) It is not clear how to implement the Evaluation function. In the initial experiments (cf. [16]), it was implemented using tables where each possible state is represented. This approach is clearly not viable in realistic environments. A more appropriate approach is the one used in ALQAAA agents where only *categories* of undesired states have to be defined, which is often much easier than to define complex reward functions.

3.2 Limitations of Experiments

The problems that the ALQAAA agents solved above can certainly be solved by other methods, but the point to be made is that we can qualitatively enhance the abilities of a reactive agent by embedding it in an ALQAAA agent. However, there are several obvious limitations to the application presented in this paper: (i) the environment is quite static (the only events that take place not caused by the agent itself are those caused by other agents), (ii) the agents have perfect models of the world,[4] (iii) the agents have perfect sensors and the outcome of an action is deterministic, and (iv) this is just a simulation

[4] Thus, the behaviour of the agents in the experiments could be regarded as anticipatory rather than quasi-anticipatory.

(the agent is neither embodied nor situated) and thus escaping the hard problems of perception and uncertainty. Moreover, only a single domain has been investigated. Future work includes evaluation of the approach in other domains to see in which types of applications it performs well and whether there any in which it is not appropriate.

References

1. P. Davidsson, E. Astor, and B. Ekdahl. A framework for autonomous agents based on the concept of anticipatory systems. In *Cybernetics and Systems '94*, pages 1427–1434. World Scientific, 1994.
2. T. Dean and M. Boddy. An analysis of time-dependent planning. In *AAAI-88*, pages 49–54, 1988.
3. M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *AAAI-90*, pages 138–144, 1990.
4. B. Ekdahl, E. Astor, and P. Davidsson. Towards anticipatory agents. In M. Wooldridge and N.R. Jennings, editors, *Intelligent Agents — Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence 890*, pages 191–202. Springer Verlag, 1995.
5. I.A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, University of Cambridge, 1992.
6. E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAI-92*, pages 809–815, 1992.
7. P.J. Gmytrasiewicz and E.H. Durfee. Rational coordination in multiagent environments through recursive modeling. *(submitted for publication)*, 1995.
8. S. Hanks and R.J. Firby. Issues and architectures for planning and execution. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 59–70, San Mateo, CA, 1990. Morgan Kaufmann.
9. S. Hanks, M. Pollack, and P. Cohen. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. Technical Report 93–06–05, Department of Computer Science and Engineering, University of Washington, 1993.
10. F.F. Ingrand, M.P. Georgeff, and A.S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, 1992.
11. D.N. Kinny and M.P. Georgeff. Commitment and effectiveness of situated agents. In *IJCAI-91*, pages 82–88, 1991.
12. M. Minsky. *The Society of Mind*. Simon and Schuster, 1986.
13. M.E. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally evaluating agent architectures. In *AAAI-90*, 1990.
14. R. Rosen. Planning, management, policies and strategies: Four fuzzy concepts. *International Journal of General Systems*, 1:245–252, 1974.
15. R. Rosen. *Anticipatory Systems – Philosophical, Mathematical and Methodological Foundations*. Pergamon Press, 1985.
16. R.S. Sutton. First results with Dyna, an integrated architecture for learning, planning and reacting. In W.T. Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control*, pages 179–189. MIT Press, 1990.
17. M.J. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
18. S. Zilberstein and S.J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *IJCAI-93*, pages 1402–1407, 1993.
19. G. Zlotkin and J.S. Rosenschein. Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domain. In *AAAI-94*, pages 432–437, 1994.