Highly reactive decision making: a game with Time

Silvia Coradeschi
IDA
Linköpings Universitet
S-581 83 Linköping, SWEDEN

Thierry Vidal
LGP, ENIT
47 avenue d'Azereix
F-65016 Tarbes cedex, FRANCE

Abstract

Real-time monitoring calls for decision making capabilities in reaction to observed events. Associative models provide efficiency by matching the observed situation to a recorded pattern equipped with an accurate decision. We rely on a *decision tree* accounting for the context and *temporal chronicles* expressing dynamic patterns. In *highly reactive* domains, i.e. when actions get as frequent as observations, the decision must anticipate the complete recognition of a pattern, comparing possible evolutions. This paper focuses on the on-line decision process, a game against Nature in the general case: a *timed game automaton* gathers the possible next steps with associated goodness values, and uses an *opportunistic* algorithm to compute a temporally expressive decision, maximizing its *utility*, i.e. the chances of "winning".

1 Introduction

Monitoring and supervision deal with dynamic systems (or *artefacts*), that evolve across time. A (human or automated) agent is in charge of *observing* (through sensors) what happens, *recognizing* the typical behaviour, and *acting* towards the artefact (through activators), for instance to put it back into a normal state, or to process some safety procedure. This on-line process needs *real-time* efficiency, and is *reactive* in the sense that actions are taken according to observations.

A model-based approach [Dvorak and Kuipers, 1989] has been introduced to deal with this problem, first computing off-line faulty models of the artefact that are used on-line as *associative models* matched with incoming observations. Next steps are predicted from the hypothetical current situation, and incoming observations are checked to confirm the hypothesis. But time is only implicit, through successive states in a dynamic decision tree, which is not expressive enough in realistic domains. Some approaches [Nejdl and Gamper, 1994] manage to attach a temporal qualification to behavioral modes, using a rich set of temporal relations. More appealing are *temporal chronicle* recognition systems [Dousson *et al.*, 1993], where temporal constraint networks are used as associative models.

But all these approaches deal with systems in which an action is the consequence of a complete and fully recognized sequence of observed events. This paper deals with an *explicit temporal* framework in *highly reactive* domains, in which actions and observations continuously respond to each other. We extend the temporal chronicle formalism to mix events representing *observations* from the artefact and the agent *actions*, and equip them with *goodness values* expressing preferences among them. We also use a decision tree such as the one designed in [Coradeschi *et al.*, 1996], but here only as a preprocessing step before decision making, to merely branch the current static context onto a subset of relevant chronicles, therefore restricting the number of candidate patterns.

The agent has now a larger choice of decisions to take at any time. The goal is not to recognize a bad situation and put the system back into a good one, but to continuously take decisions that anticipate bad situations and push the system into better ones, which is basically a game playing process. For that we rely on the *timed game automaton* model [Asarin *et al.*, 1995] that is best suited for continuously changing systems. Next transitions are computed from the set of candidate patterns, inheriting corresponding *goodness values*. Then our new algorithm PLAY-AUTOMAT chooses a transition to select and a correct time to do so, using a *least-commitment* and *opportunistic* strategy, and following a formal decision policy based on a definition of the *utility* of each possible decision: we prove that our process is *optimal* in that it always takes the decision with highest utility.

Section 2 recalls the basic models and global architecture described in [Coradeschi and Vidal, 1998]. Our approach is illustrated through a specific example in the area of one-to-one aircraft combat in Section 3. Then Section 4 focuses on the very contribution of this paper, namely the new algorithm for on-line decision making.

2 Basic models and global process

2.1 The basic decision-tree approach

In [Coradeschi *et al.*, 1996], the agent is equipped with a *context* (i.e. a set of propositions describing the current

state of the world), and a decision tree. To each leaf of the tree is attached a decision (atomic action or sequence of actions). At each step of the simulation process the context is updated with new events received and interpreted. Then the decision tree is visited down, propositions in the context being matched with the conditions appearing at each node, until a leaf is reached. Each action has a priority value that changes dynamically, so that in the cases where multiple leaves are applicable and actions are mutually exclusive, the best one is selected. It is then sent to the simulator, which will update the context, and so forth.

This mechanism is highly reactive and efficient, and agent behaviours are easy to specify and test. It is however difficult to code in it reactions to sequences of temporally related events.

2.2 Possible evolutions as chronicles

In the temporal system IxTeT [Dousson *et al.*, 1993] used for dynamic situation assessment, temporal evolutions are taken into account in the shape of *chronicles*, that are *Temporal Constraint Networks* [Schwalb and Dechter, 1997] on which classical constraint propagation techniques can be run: time-points represent instantaneous changes or begin/end points of intervals of time over which a fluent is true, and constraints between them are precedences labelled by arithmetic intervals of possible values, allowing to express dates of events and durations of fluents. Then chronicles are matched with incoming events, dynamically maintaining the set of candidates. As soon as a chronicle is fully *recognized*, an action written in the chronicle description is triggered.

This model is well-suited for dynamic applications with temporally expressive behaviours like nuclear plant or gas turbine monitoring [Milne *et al.*, 1994], where supervision is the key word. As an associative model, it provides high on-line efficiency; anyway, two shortcomings compelled us to improve it somehow.

- Reactivity can be considered as being weak in IxTeT, since an action only follows an ordered set of observations. Therefore we chose to extend the initial formalism to mix events representing both *observations* from the artefact and the agent *actions*, which in turn enforces a corresponding distinction between two types of constraints between events, as in [Vidal and Fargier, 1997]. A numerical constraint between e_1 and e_2 (with e_1 *before* e_2) will be said to be *controllable* iff e_2 is an action, and *contingent* iff e_2 is an observation.
- The second restriction is in the classical strict distinction between normal and abnormal behaviours: the agent here has a large choice of actions at any time that can push the system into various "more or less good" situations. Therefore we chose to extend the IxTeT approach by adding to each chronicle a *goodness* value in the range [-1,1], -1 meaning the worst possible case (e.g. breakdown) at the chronicle completion, 1 a behaviour that fully entails the system specifications, 0 a situation that keeps balanced between eventual failure or success.

2.3 Timed game automata

The model we present here is inspired by recent advances [Asarin *et al.*, 1995] on *timed automata* models [Alur and Dill, 1994], used for describing the dynamic behaviour of a system. It consists in equipping a finite-state automaton with time, allowing to consider cases in which a system can remain in a state during some time t before taking the next transition. This is made by adding continuous variables called *clocks* that are reset when some transitions are taken, then grow uniformly until they are checked on a later transition through some condition (*guard*) that must be true for enabling it.

Such tools are well-suited for *continuous real-time games*: for each player, transitions are either *activated* or *received*, and some states are designated as *winning* ones. That extends the discrete game approach [Pearl, 1984], with the following pros: (1) there are no "turns" and the adversary need not wait for the player's next move, and (2) each player not only chooses a transition, but also the delay to wait before taking it.

This is especially relevant for controlling reactive systems in which one has to "play against Nature", the goal being to *synthesize* a "safe" controller, i.e. add conditions to compel the automaton to reach winning states for the agent. For our purpose we only need a restricted model: we do not need to build a complete automaton but we just compute the next states, transferring the goodness values from corresponding chronicles so as to compare them and choose the "best" one. That leads to the following restricted game automata definition.

Definition 2.1 (Short-term Game Automata)
$\mathcal{A} = (\mathrm{Q}, \mathrm{w}, \mathrm{Z}, \Sigma, \mathrm{T}, \mathrm{q_0})$ *is a game automaton where*
- Q *is the discrete set of states,*
- $q_0 \in \mathrm{Q}$ *is the initial (current) state,*
- $\mathrm{w} : \mathrm{Q} \rightarrow [-1, 1]$ *adds goodness values to states,*
- $\mathrm{Z} = (\Re^+)^{\mathrm{d}}$ *is the clock space,* $\mathtt{H}(\mathrm{Z})$ *is the set of condition relations over clocks (see [Asarin* et al., *1995] for details), and* $\mathtt{R}(\mathrm{Z})$ *is the set of reset functions on clocks,*
- $\Sigma = \Sigma_b \times \Sigma_a \times \{\mathbf{wait}\}$ *is the input alphabet (***b/a** *events plus the specific event* **wait**, *see 4.2),*
- $\mathrm{T} = \mathrm{T_b} \times \mathrm{T_a} \subseteq \mathrm{Q}^2 \times \Sigma \times \mathtt{H}(\mathrm{Z}) \times \mathtt{R}(\mathrm{Z})$ *is the set of transitions of the form* $\tau = < q, q', \sigma, g, r >$ *where*
  $(q, q') \in \mathrm{Q}^2, \sigma \in \Sigma, \mathrm{g} \in \mathtt{H}(\mathrm{Z}), \mathrm{r} \in \mathtt{R}(\mathrm{Z})$,
  $\tau \in \mathrm{T_a}$ *is* activated $\mathtt{iff}\ \sigma \in \Sigma_a$,
  $\tau \in \mathrm{T_b}$ *is* received $\mathtt{iff}\ \sigma \in \Sigma_b$.

2.4 Global architecture

We have already presented [Coradeschi and Vidal, 1998] our associative model mixing a decision tree and chronicles, simply replacing actions in leaves by sets of chronicles, as a simple way to add context handling to the chronicle model. Then our global process will be to (1) determine the set of *active* candidate chronicles visiting down the decision tree, (2) build the corresponding automaton, extracting from the chronicles the possible

next states and transitions to them, and (3) select the best decision to make NOW by "playing" the automaton. This is illustrated through figure 1.

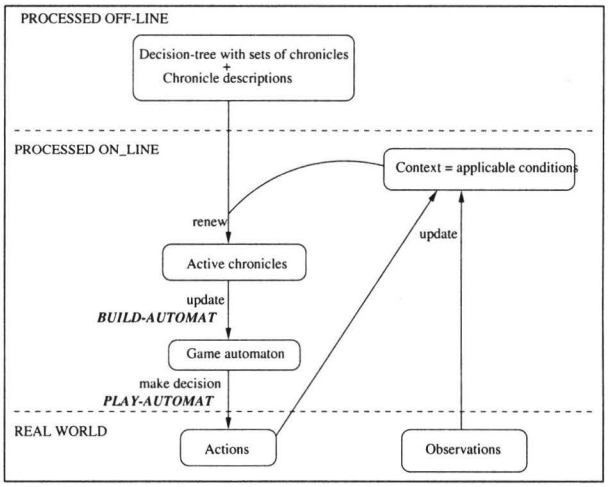

Figure 1: The new proposed architecture

The global algorithm and its two first steps will not be reported here (see [Coradeschi and Vidal, 1998]), but the following example should give the reader an insight into it. Then we will focus on the decision-making step (3), namely the algorithm PLAY-AUTOMAT.

3 An example in the air-combat domain

The air-combat domain is a highly reactive domain where decisions are made under real-time constraints. An automated pilot should be able to predict his opponent's next moves and select the action that minimizes possible threats and maximizes chances of success. Predicting other agents behaviours is a hard task but some typical patterns have been developed by the military to help in identifying manoeuvres of the opponents.

We have built a simplified example of a one-to-one beyond visual range combat situation (the aircrafts can see each other just with board instrumentation). Figure 2 shows the chronicles corresponding to most plausible typical patterns, where events labelled with **a** are the automated pilot own actions and the ones with **b** are the opponent observed actions. We start with **a** and **b** flying towards each other, and **a** sees **b** moving right. Two possible guesses for **a** are that **b** will continue escaping or turn back for an intercept. Then **a** can move left (evolution Q_1), making **b** moving left as well. The resulting situation gives no special advantage for any of the pilots. Otherwise **a** may accelerate, then both pilots turn left for an intercept (Q_2), which would put **a** into a better situation as he can more easily attack on the side, or he may observe **b** escaping by moving right (Q_3), which is even better. Q_4 is also a good situation as **a** does not do anything and **b** escapes. But not doing anything might as well be bad, if **b** moves left to intercept (Q_5). Precise delays (here in seconds) are also added to the constraints.

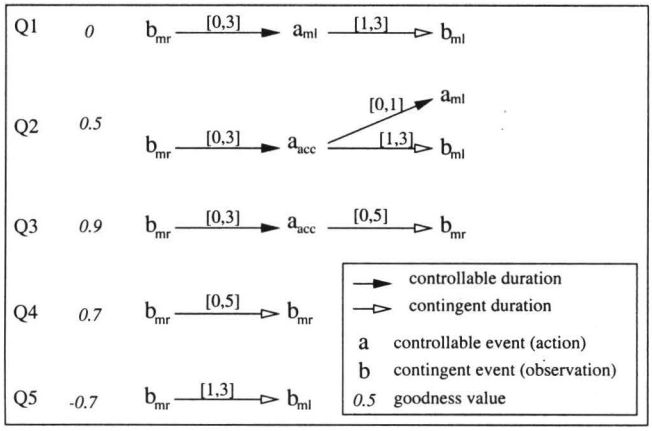

Figure 2: The temporal chronicles of the example

Then figure 3 shows the result of step (2), the algorithm `Build-automat`. One should notice the correspondence between durations of controllable/contingent constraints in chronicles and guards on activated/received transitions in the automaton. Receiving the event $\mathbf{b}_{mr}$ puts the pilot into a state in which there are four posible next events $\mathbf{e}_{next}$ given by the five chronicle candidates of figure 2. For each one a new state is added together with the transition to this state, labeled with $\mathbf{e}_{next}$, computing the guard from the corresponding chronicle. Goodness values are transferred as well, with here the case where an event ($\mathbf{a}_{acc}$) belongs to two different chronicles, so one keeps the min of the goodness values.

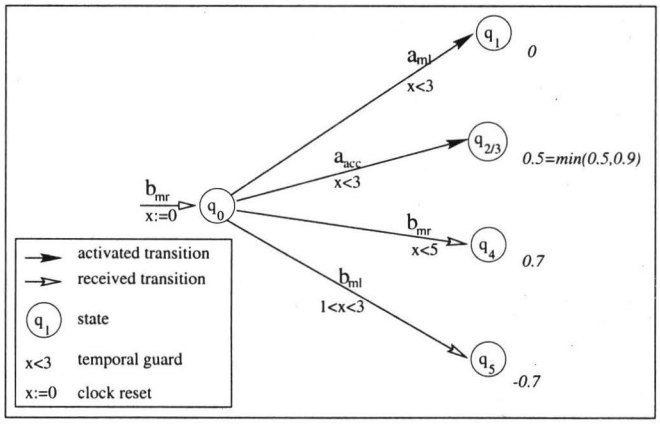

Figure 3: The game automaton of the example

4 On-line decision making

4.1 Decision policy: basic definitions

Before presenting the algorithm that computes an accurate decision from the built automaton, wes first define

our decision policy, in the spirit of studies carried on in qualitative decision theory (see e.g. [Boutilier, 1994]). Goodness values provide preferences over consequences of the decisions, but a decision might raise distinct consequences. In this uncertain framework the *utility* of a decision must be defined as a function over the utilities (the goodness values) of all the possible consequences. For that we rely on a choice criterion similar to the classical pessimistic/cautious Wald criterion, that suits well our illustrative application area. Of course in other domains less strict criteria balancing between the risk and the expected outcome may be preferred.

Definition 4.1 *The* utility *of a decision* U*(δ) is the lowest utility of all its possible consequences, i.e. the lowest goodness value of all the possible states in which the system might get after the decision δ is taken.*

The decision strategy amounts to choose the decision with highest utility, which is consistent with min-max game strategies [Pearl, 1984]. We will now give our algorithm and prove that it obeys the given criterion, but before that we need some additional definitions.

Definition 4.2 *A selected transition τ is said to be* ensured *iff the system cannot take another one.*
On the contrary, τ is said to be threatened *by τ' (the* threat*) if the system might take τ' instead of τ (obviously τ' can only be a received transition).*
We use the same terms for the corresponding decision δ of selecting the transition τ.

4.2 The game-based decision procedure

Our procedure has been inspired by classical *controller synthesis* algorithms [Asarin *et al.*, 1995], but highly simplified in our case of "one step ahead" automata with goodness values. One needs first sorting the next transitions starting from the one with highest goodness value, and then consider them one by one.

1) If the current best transition τ is a **received** one, the decision is to wait for its labelling event to occur. We watch out if there is a worse received transition τ' possibly threatening τ by occurring meanwhile (test `lower(g(τ')) ≤ upper(g(τ))`). Ensuring τ amounts to deciding to wait only while τ' cannot occur: τ can hence be ensured iff `lower(g(τ)) < lower(g(τ'))`. If not, this selection is forgotten and one tries next item in RANKED (`Goto` *0*). Otherwise the waiting `delay` is set to `lower(g(τ'))`, and a new transition is added, labelled by the special event **wait**, that will be taken for sure just *before* the `delay` has elapsed (the guard therefore exactly equals `delay`$^-$, which means the atomic duration immediately lower than `delay`). Then there are two possibilities; in the worst case, nothing happens, the transition **wait** is taken after `delay`$^-$ has elapsed, and a new automaton accounting for this elapsed time is recomputed in the next stage of the global algorithm; or in the lucky case where the event labelling τ occurs before the `delay` has elapsed, then τ is taken.

The algorithm has been slightly simplified for the sake of clarity: it should take into account cases with several threats: the delay to wait is actually the min of the corresponding delays.

Last, when there is no threat τ', the decision is just to wait for τ to be taken, but actually no more than `MaxDelay`, which corresponds to the time after which a better activated transition that had been put aside may become releasable, which is explained herebelow.

2) In the case where τ is an **activated** transition, things get simpler, but τ still needs to be ensured. If the action can be released now (`lower(g(τ)) = 0`), then it is released ($\sigma(\tau)$ is the action label of τ). If not, the test about a possible threat is now on the value `lower(g(τ))` instead of `upper(g(τ))` (τ being an activated transition it can be released as soon as `lower(g(τ))` has elapsed). Here again, if τ cannot be ensured, then this selection is put aside and the next one tried. But in the case where the decision ends up being **wait**, as soon as the action $\sigma(\tau)$ becomes releasable, it will not be threatened anymore (`lower(g(τ))` equalling 0), and it should be reconsidered as a possible decision. For that we use `MaxDelay` that keeps track of the min of the times after which an action becomes releasable, so as to restart the algorithm at that time.

```
PLAY-AUTOMAT(𝒜):
    RANKED := set of next transitions ordered from the
                highest to the lowest goodness value
    MaxDelay := +∞
0:  τ := pop(RANKED)
    If τ ∈ T_b then
      | if ∃τ'∈ RANKED s.t. τ' ∈ T_b
      |          and lower(g(τ')) ≤ upper(g(τ)) then
      |   | delay := lower(g(τ'))
      |   | if delay leq lower(g(τ)) then Goto 0
      |   | else
      |   |   | add a new state q_wait
      |   |   | add a new transition:
      |   |   |     τ_wait =< q_0, q_wait, wait, {delay}, ∅ >
      |   |   | DECISION := (wait, delay⁻)
      | else DECISION := (wait, MaxDelay)
    Else
      | if lower(g(τ)) = 0 then DECISION := (σ(τ), NOW)
      | else   if ∃τ'∈ RANKED s.t. τ' ∈ T_b
      |          and lower(g(τ')) ≤ lower(g(τ)) then
      |       | MaxDelay := min(MaxDelay, lower(g(τ))
      |       | Goto 0
      |   else DECISION := (σ(τ), lower(g(τ)))
    send DECISION to the execution manager.
```

This algorithm is illustrated through the example of figure 3. In the first iteration, q_4 is the best transition (maximal goodness value 0.7), but it is threatened by the transition to q_5. Ensuring the selected transition amounts to wait no longer than 1. The new **wait** transition added to the automaton appears in figure 4. We show there a possible improvement of the algorithm (not developped here for the sake of clarity) where one not only adds this transition, but at the same stage computes the next states from q_{wait}, which should be the same as

in q_0, with recomputed guards according to the `delay`. The two activated transitions from q_0 are in dashed lines because they will not be taken at this stage, trying to opportunistically let b_{mr} occur first. Next, falling into q_{wait}, the choice b_{mr} would be forgotten since it can no longer be ensured, and the next best transition a_{acc} would be selected since it can be released at once. This improvement, which amounts to consider two successive decision steps in one stage, would end up with an expressive decision sounding like *wait no longer than one second, then accelerate*, which of course could be cancelled by the early fortunate occurrence of b_{mr}.

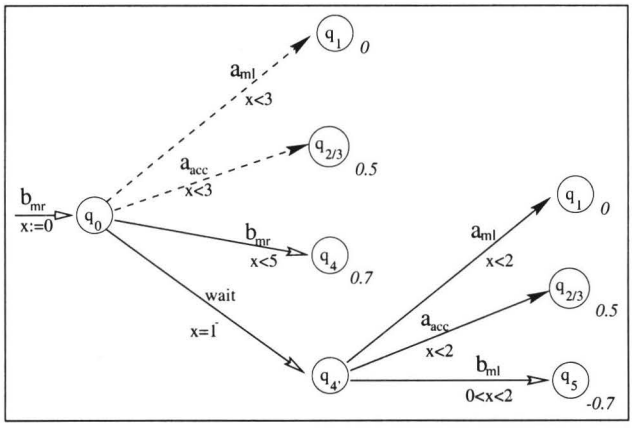

Figure 4: The wait policy

4.3 Decision making algorithm properties

Our algorithm is a *least commitment* strategy: one always prefers waiting if this may lead to a better situation and is not threatened by a worse one. It is as well an *opportunistic* process: actions that were put aside because they could not be ensured are considered again when they become releasable, and waiting instead of releasing a "rather good" action may allow to receive a better event, such as in our example. Last, our algorithm is *definite* in the sense that a decision will always be taken (provided the automaton is not empty): even a **wait** decision ensures the system evolution either when a received event occurs, or when `MaxDelay` has elapsed and an action can then be released.

About algorithmic complexity, it gets low first thanks to our associative models (see [Coradeschi *et al.*, 1996] and [Dousson *et al.*, 1993]), but also in the algorithm PLAY-AUTOMAT: the automaton is bounded in *depth* (one never computes a complete automaton but only needs to consider the next step thanks to the goodness values), and also in *breadth* (the number of alternative transitions to consider is lower than the maximum number of chronicles in a given context, i.e. in a leaf of the decision tree). In fact the global complexity only depends upon the size of the decision tree and the maximum number of chronicles in each leaf, i.e. the size of the expertise knowledge base, which is constant and known.

The key point is to prove the *optimality*, i.e. that one always gets the best decision according to the given policy (of course the decision taken will only be the best according to the given goodness values, hence it highly depends on how accurate and realistic the associative model built off-line is). For that purpose, we check that in each case the decision with highest utility is taken.

1. If the transition τ with highest goodness value γ is an activated one (the decision δ will be an action):
   (a) if δ is not threatened: the action will necessarily be released and δ has the highest utility.
   (b) if δ is threatened by τ^t leading to a state with goodness value γ^t: then $U(\delta)=\gamma^t$.
      i. if there exists δ' not threatened leading to a state with goodness value γ' s.t. $\gamma > \gamma' > \gamma^t$: the algorithm will take the decision δ' with $U(\delta')= \gamma' > U(\delta)$.
      ii. if there exists δ' threatened: the reasoning process can be recursively applied as in 1.(b).
      iii. if no other satisfactory decision is found: the algorithm only waits until some action becomes releasable (thanks to `MaxDelay`), hence all the other possibly better decisions remain available at the next stage.
2. If the transition τ with highest goodness value γ is a received one (the decision δ will be to wait):
   (a) if δ is not threatened: it is ensured hence the corresponding state will be reached, and δ has the highest utility.
   (b) if δ is threatened by an event that may occur after a certain delay: the system will wait for this delay, in an opportunistic way, which might hopefully lead the system into the state reached by τ (which means highest utility), or put it into the following situation (c).
   (c) if δ is threatened by an event that may occur immediately: δ has the same utility as the threat, and hence is not better and might be forgotten, as the algorithm does.

5 Discussion and conclusion

A main strength of our approach is to use different models that are best suited for each of the requirements of highly reactive monitoring: a decision-tree for the *context*, chronicles for the *temporal evolutions*, and a game automaton for the *predictive* decision making process. It is straightforward to integrate those models and make them work together in a smooth way.

Our approach can be compared to [Tambe and Rosenbloom, 1996], where opponent actions are considered while making dynamic decisions. There a single interpretation of actions and observations is used. Our approach is more general since actions and observations may be of any kind, making it fit the more general area of operator/artefact reactive loop. In the TIGER gas turbine monitoring project [Milne *et al.*, 1994], three models are

mixed in a similar way as ours: decision trees are replaced by compiled rule bases, and a classical chronicle recognition mechanism feeds a model-based diagnosis system. But we do more than mere supervision, getting a highly reactive decision making system processing hypothetical reasoning on the future instead of comparative reasoning on the past. This compelled us to improve the chronicle structure to incorporate actions in it. In a related game-based military application [Katz and Butler, 1994], decision is also dynamically made, but thanks to a discrete game model. Besides, classical heuristic techniques are used, exploring down the tree in a lookahead simulation, which we avoid by directly inheriting goodness values from our associative model.

One shortcoming of our approach is the classical expertise acquisition problem: *uncompleteness* of such an expertise is always to be feared. Anyway, our architecture can be made robust to unexpected events as well, thanks to some additional features that have not been presented here (see [Coradeschi and Vidal, 1998]).

As a matter of conclusion, our approach is relevant in highly reactive real-time monitoring applications with complex temporal constraints. It brings out expressive and accurate dynamic decision making. We strongly believe that mixing symbolic models from the artificial intelligence community and control models built by theoretical computer scientists, as we did, might help making substantial advances in that field.

Acknowledgments

Thierry Vidal has been supported by the *Excellence Center for Computer Science and Systems Engineering (ECSEL)* in Linköping. Silvia Coradeschi has been supported by the Wallenberg Foundation project *Information Technology for Autonomous Aircraft*. The authors are also grateful to Dan Strömberg (Swedish National Defense research center, Linköping) and Göran Petterson (SAAB Military Aircraft, Linköping) for useful comments and discussions that inpired the simplified application example used in the paper.

References

[Alur and Dill, 1994] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Asarin *et al.*, 1995] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II, LNCS 999*. Springer Verlag, 1995.

[Boutilier, 1994] C. Boutilier. Toward a logic for qualitative decision theory. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR-94), Bonn (Germany)*, pages 75–86. Morgan Kaufmann, 1994.

[Coradeschi and Vidal, 1998] S. Coradeschi and T. Vidal. Accounting for temporal evolutions in highly reactive decision-making. In *Proceedings of the 5th International Workshop on Temporal Representation and Reasoning (TIME-98)*, Sannibel Island (FL), 1998.

[Coradeschi *et al.*, 1996] S. Coradeschi, L. Karlsson, and A. Törne. Intelligent agents for aircraft combat simulation. In *Proceedings of the 6th Conference on Computer Generated Forces and Behavioral Representation*, pages 23–26, Orlando (FL, USA), 1996.

[Dousson *et al.*, 1993] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: representation and algorithms. In *Proceedings of the 13th International Joint Conference on A.I. (IJCAI-93)*, Chambéry (France), 1993.

[Dvorak and Kuipers, 1989] D. Dvorak and B. Kuipers. Model-based monitoring of dynamic systems. In *Proceedings of the 11th International Joint Conference on A.I. (IJCAI-89)*, pages 1238–1243, Detroit (MI, USA), 1989.

[Katz and Butler, 1994] A. Katz and B. Butler. "Game Commander"- Applying an architecture of game theory and tree lookahead to the command and control process. In C. Backström and E. Sandewall, editors, *Proceedings of the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville (FL, USA), 1994.

[Milne *et al.*, 1994] A. Milne, C. Nicol, M. Ghallab, L. Travé-Massuyès, K. Bousson, C. Dousson, J. Quevedo, J. Aguilar, and A. Guasch. TIGER: real-time situation assessment of dynamic systems. *Intelligent Systems Engineering*, pages 103–124, 1994.

[Nejdl and Gamper, 1994] W. Nejdl and J. Gamper. Harnessing the power of temporal abstractions in model-based diagnosis of dynamic systems. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 667–671, Amsterdam (Netherlands), 1994.

[Pearl, 1984] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley, Reading (Mass, USA), 1984.

[Schwalb and Dechter, 1997] E. Schwalb and R Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93:29–61, 1997.

[Tambe and Rosenbloom, 1996] M. Tambe and P.S. Rosenbloom. Architectures for agents that track other agents in multi-agent worlds. In C. Backström and E. Sandewall, editors, *Agents, Theories, Architectures, and Languages (ATAL-95)*. Springer Verlag Lecture Notes in Artificial Intelligence 1037, 1996.

[Vidal and Fargier, 1997] T. Vidal and H. Fargier. Contingent durations in temporal CSPs: from consistency to controllabilities. In *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME-97)*, pages 78–85, Daytona Beach (FL, USA), 1997.