Accounting for Temporal Evolutions in Highly Reactive Decision-making

Silvia Coradeschi
IDA
Linköpings Universitet
S-581 83 Linköping, Sweden
e-mail: silco@ida.liu.se

Thierry Vidal
LCPSI, ENIT
47 avenue d'Azereix
F-65016 Tarbes cedex, France
e-mail: thierry@enit.fr

Publication and copyright

This paper was presented at the TIME-98 international workshop, May 16-17, 1998, Sanibel Island (Fl, USA), the proceedings of which were published by the IEEE.

Abstract

Applications such as aircraft combat simulation require both dynamic supervision and real-time decision-making : actions and observations interact in a reactive way, and durations between two events must be accounted for. We start from a decision-tree *model, which owns strong context handling capabilities, but provides only static decision-making. We improve it by matching possible* temporal chronicles*, accounting for dynamic evolutions, to the context. A decision in this framework is activated upon complete recognition of such a chronicle, whereas our highly reactive application domain requires anticipated decisions, comparing possible evolutions to take the best decision in real-time. We choose to dynamically compute a* timed game automaton *synthesizing the predicted possible next steps, and propose an algorithm that computes the best decision from this simple controller model.*

1 Introduction

We intend to propose a decisional architecture integrating multiple models, aimed at decision-making in real-time and highly reactive environments. We have chosen to base this theroretical study on a simplified toy example inspired by the application domain of one-to-one aircraft combats, and that we present in the next section. Then section three separately introduces three distinct classical approaches that can only address different stages of the problem, namely *decision trees* that are used to take static decisions, efficiently accounting for the context, *temporal chronicles* that are able to synthesize and efficiently assess on-line complex evolutions, and *timed game automata* that are best suited for controlling a dynamic situation, determining the best next transition to activate. Improvement and/or simplification of those models to better fit our application are described as well. In section four lies the very contribution of this paper: to meet the ambitious requirements of high temporal expressiveness, real-time reactive behaviour and high quality of the decisions taken, we integrate those three models into a global real-time architecture, which presentation is upported by sketches of algorithms. Finally in section five a comparative discussion summarizes the strengths and originality of our approach.

2 The application domain

The air-combat domain is a highly reactive domain where decisions are made under real-time constraints and with incomplete knowledge of the current dynamically evolving situation. A main source of uncertainty is the lack of knowledge about the intentions and possible next moves of the opponent. To make the right decision 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 in this kind of highly dynamic and adversarial domain, but some typical patterns have been developed by the military to help in identifying manoeuvres of the opponents.

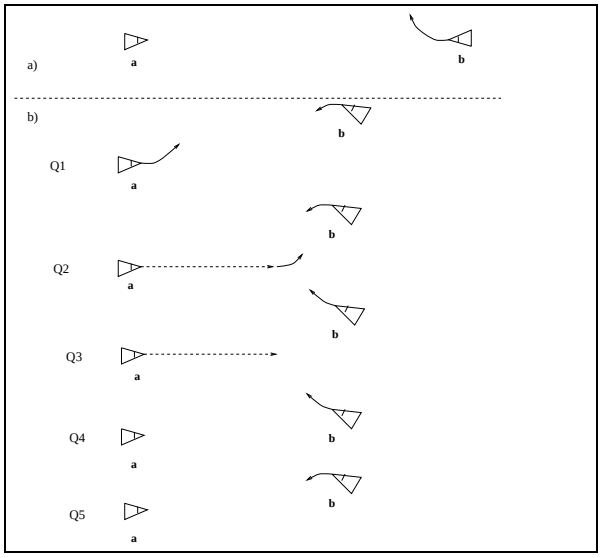

Figure 1: Example of typical patterns in a one-to-one beyond visual range combat

An example of a one-to-one beyond visual range combat situation (the aircrafts can see each other just with board instrumentation) is presented in figure 1 (please notice that this example is mainly illustrative and does not correspond to actual military doctrine). In figure 1 a), the automated pilot (aircraft **a**) sees the adverse aircraft **b** turning to the right. Two possible guesses for pilot **a** are that the pilot **b** will continue increasing his distance and escape or that he will turn back for an intercept. The possible evolutions of the situation depend on the actions of both **b** and **a**. The pilot **a** can turn toward pilot **b**, can continue straight or can continue straight and accelerate. In figure 1 b) are considered possible evolutions of the situation. Of course these are not all the possible evolutions, but are the most plausible patterns corresponding to typical cases that arise in real situations.

Let us consider it in more details. In evolution Q_1, **a** turns to the left and **b** turns back to the left. The resulting situation is similar to the initial one with no special advantage for any of the pilots, as far as the distance still lies in the same range. In evolution Q_2, **a** accelerates before turning to the left and **b** turns back to the left. Here there is an advantage for pilot **a** as he can more easily attack on the side. In evolution Q_3, **a** accelerates and **b** continues to escape. This is a very good situation as the pilot **a** does not need to fight, and at the same time is in a perfect position to attack **b** from behind. In evolution Q_4, **a** does not do anything and **b** continues to escape. This is also a good situation as the pilot **a** does not need to fight. In evolution Q_5, **a** does not do anything, but **b** turns to the left to intercept. This is a bad situation as the pilot **a** will be likely attacked on the side.

3 Technical background and representation issues

3.1 The current decision-tree approach

In the simulation model developed in [4], each pilot is equipped with what we will call here a *context* (i.e. information about the current state of the world), and a decision-tree. The latter specifies the behaviors of the agent according to contextual conditions, and consists of a hierarchy of decisions with one decision for each node. At every step of the simulation the context is updated with information received from the simulator and interpreted. Then the decision-tree is visited down to one (or more) applicable leaf(ves), which results in a list of possible actions. The actions in the list are the actions that the agent will consider performing. Each action is associated with a priority value that changes dynamically. To this list the actions and the activities still in progress are added. The agent then selects the actions to actually execute and send them to the simulator, which will update the context. Then we're back at the top of this global decision loop. Some of the actions in the list are mutually exclusive as they use the same resource and in this case the agent selects those he will actually perform depending on their priority values. Some actions can be performed concurrently and in this case the agent performs them in parallel. The agent can also decide to perform sequences of actions. However the agent is not committed to complete the sequence of actions as the sequence can be interrupted if an action with higher priority is selected.

We give in figure 2 an example of a decision-tree that has been designed for an interceptor aircraft aimed at protecting the area from the enemy (see [4] for details). The interceptor attacks if there is a target and enough fuel (*and tgt1 selected, percentage fuel left* $>$ *15.0*). In the case of an attack the interceptor moves along a path that allows to come near to the opponent, but without entering its firing area (**gimbal-limit**). When the conditions for firing are favorable it fires an active missile (**fire-active-ms**).

The decision-tree is a highly reactive and rapid decision mechanism. Behaviors of pilots are easy to

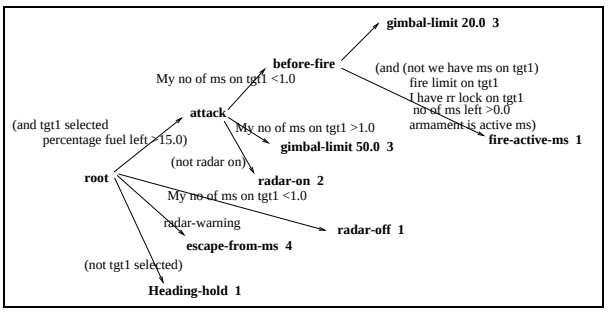

Figure 2: A decision tree for an interceptor aircraft that is supposed to protect the area from the enemy.

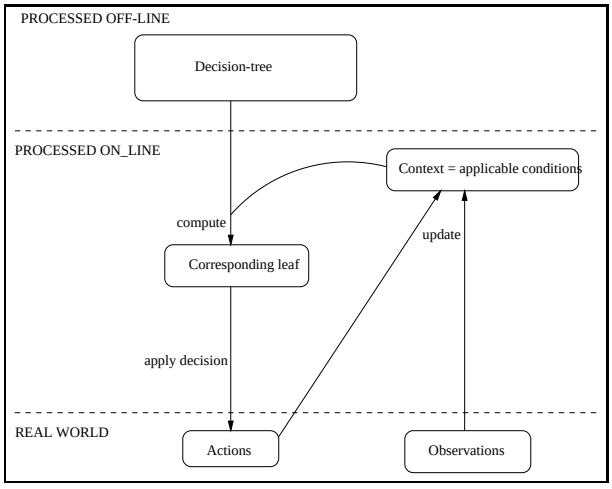

Figure 3: The decision-tree architecture

specify and test. It is however difficult to code in it reactions to sequences of temporally related events. A decision of the pilot can indeed depend on sequence of temporally related events: for instance if the pilot changes course and shortly after so does the opponent this can be a sign that the opponent has seen him. The figure 3 shows a simplified view of the decision-tree approach that is sufficient for our purpose, and that we plan to improve adding explicit temporal management techniques, like the one described in the following subsection.

3.2 Handling possible evolutions as chronicles

Taking into account temporal evolutions for dynamic situation assessment has been considered in [5], based on the temporal system IxTeT. Typical patterns describing situations likely to arise in the application domain are represented in terms of *chronicles*. A chronicle encompasses one possible behaviour of the supervised real system, often distinguishing between normal and abnormal behaviours, so as to enforce diagnosis capabilities. The chosen temporal formalism for one chronicle rely on a set of time-points corresponding to instantaneous changes (e.g. a valve moves from 'open' to 'closed'), or to begin/end points of intervals of time over which a fluent is true (e.g. the temperature remains low between two specified times). One eventually gets only time-points related by precedence (symbolic) constraints, to which one can add imprecise numerical constraints, i.e. dates of events and durations of fluents by ways of simple arithmetic intervals of possible values (e.g. a flow output is observed between 1 and 2 seconds after the valve has been opened). The resulting graph is nothing but a *Temporal Constraint Network* [10] on which classical constraint propagation techniques can be run. Those chronicles can be seen as an *associative* model, since they are computed off-line, and then matched on-line with the incoming observed events, checking that the delays between them satisfy the chronicle constraints. Only chronicles that are compatible are dynamically maintained in a set of candidates. As soon as a chronicle is fully matched by real events, it is *recognized*, and an action written in the chronicle description is triggered, which can be a message to the operator, the derivation of a new complex event, an emergency automatic reaction, etc.

This explicit temporal model is especially well-suited for dynamic applications like nuclear plant or gas turbine monitoring [8], where supervision is the key word. As an associative models, it provides high on-line efficiency, satisfying the real-time requirement. Anyway, a first shortcoming to be pointed out is that context handling with IxTeT is limited in the sense that one must add a context condition (in the shape of an assertion) directly in the chronicle description. This one-to-one link makes it difficult on one hand to specify a chronicle valid in various alternative contexts, and conversely having various chronicles valid in the same context can only be represented through the explicit addition of this context in all the chronicles. Hence one would like a more flexible approach, as decision trees give it.

The second restriction is about reactivity capabilities, which can be considered as being weak in IxTeT. In our framework, the pilot cannot wait until complete recognition of one chronicle, but he must anticipate it. At the reasoning level, this amounts to higher prediction, not only focusing on the likely next steps, but processing some kind of *hypothetical* reasoning considering the whole set of chronicle candidates to select the best decision to make NOW. This suggests a continuous game-like strategy, for which one should appeal to

more specific techniques, like the one described in the following subsection.

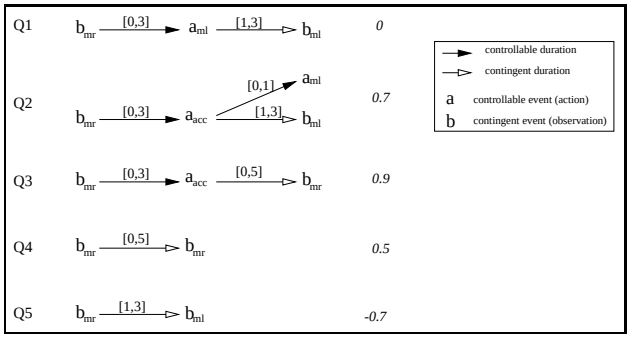

Figure 4: The temporal chronicles corresponding to the five possible scenarios

But before getting there, the previous remarks call for an extension of the chronicle formalism to mix events representing both observations from the opponent (identified by **b** letters) and own actions (**a** letters). For instance, $\mathbf{b}_{ml}$ [resp. $\mathbf{b}_{mr}$] means that the adverse aircraft moves left [resp. right], or $\mathbf{a}_{acc}$ means the pilot is accelerating. This enforces a corresponding distinction between two types of constraints between events. An arrow represent a symbolic precedence between two events, valuated by a numerical interval of possible durations. Some of them are said to be *controllable* (those leading to an event **a**), which means the pilot can decide the effective duration at execution time, while the others are said to be *contingent* (those leading to an event **b**), which means that the actual duration will be given by the external world (for a more formal definition of this ontological distinction and more theoretical property analysis, that are beyond the scope of this paper, see [12]).

We also add to each chronicle a *goodness* value that will be useful for the decision-making process. The range of values is [-1,1], a value of -1 meaning the pilot "loses" at the chronicle completion, 1 meaning that he "wins", 0 that the situation between the two aircraft remains balanced. Any intermediate value indicates how better or worse the situation gets at the end of the chronicle. Applying this representation scheme to our example raises the five temporal chronicles given in figure 4, in which temporal values are added (the unit of time being the second).

3.3 Real-time decision-making in game automata

The model we present here is inspired by recent advances [2] on *timed automata* models [1], 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 making the next transition. This is made by adding continuous variables called *clocks* that are reset when some transitions are taken, then grow uniformly when the automaton is in some state, and can be tested on later transitions through conditions (*guards*) that must be true for enabling the transition.

In [2], it is argued that such tools are well-suited for addressing *continuous real-time games*, where transitions are divided in two groups depending on which of the two players control it, and some states are designated as *winning* for one of the players. The strategy for each player is to select those controlled transitions leading to winning states. This is an extension of the classical discrete game approach (see e.g. [9]), with the following advantages: (1) there are no "turns" and the adversary need not wait for the player's next move, and (2) each player not only chooses between alternative transitions, but also between *waiting* some time or not before taking it.

This is especially interesting for controlling reactive systems in which one player is the "nature" (the *environment*). The other player (the *controller*) has control over some of the transitions. The referenced work [2] gives a complete analysis of how to *synthesize* a "safe" controller, i.e. how to add conditions to enforce the automaton only to reach winning states.

For our purpose we will only need some restricted view of this model. We extract from all the current possible chronicles the next possible events (observations or actions) that can arise. Then we wish to translate this into transitions in the automata to next states. The two kinds of constraints in the chronicle (controllable or contingent) should be translated into the two kinds of transitions appearing in the game automaton model, and similarly intervals of possible durations before the event occurrence into clock resets and guards in the automaton.

Automata are basically *simulation* models. But since we use an associative chronicle model to get precomputed predictions on the possible evolutions, we do not need here to process a complete simulation : we simply transfer the goodness values from the chronicles to the corresponding next states, and looking for the "best" transition that should be taken is merely a comparative process between those next possible transitions (in a discrete game approach, it would mean we do not need to search through the tree since we directly get min-max values on the first level nodes). In other words, we do not want to check in advance that

the whole game is "safe" (which would mean ensure the pilot will always win, which is not realistic), but simply heuristically determine at each step the safer next action he can make. So we define the following restricted game automaton model that fits well our purpose:

Definition 3.1 (Short-term Game Automata) $\mathcal{A} = (\mathrm{Q}, \mathrm{w}, \mathrm{Z}, \Sigma, \mathrm{T}, q_0)$ *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,*
- $\mathrm{Z} = (\Re^+)^d$ *is the clock space,* $\mathtt{H}$ *is the set of condition relations over clocks (see [2] for details), and* $\mathtt{R}$ *is the set of reset functions on clocks,*
- $\Sigma = \Sigma_b \times \Sigma_a$ *is the input alphabet (***b/a** *events),*
- $\mathrm{T} = \mathrm{T}_b \times \mathrm{T}_a \subseteq \mathrm{Q}^2 \times \Sigma \times \mathtt{H}(\mathrm{Z}) \times \mathtt{R}(\mathrm{Z})$ *is the set of transitions of the form* $\tau = <q, q', \sigma, g, r>$ *where*
  - $(q, q') \in \mathrm{Q}^2, \sigma \in \Sigma, g \in \mathtt{H}(\mathrm{Z}), r \in \mathtt{R}(\mathrm{Z})$,
  - $\tau \in \mathrm{T}_a$ *is* activated `iff` $\sigma \in \Sigma_a$,
  - $\tau \in \mathrm{T}_b$ *is* received `iff` $\sigma \in \Sigma_b$.

We give in the next section sketches of algorithms for building the automaton and checking it, but we need first to thoroughly define our global architecture integrating the three models presented so far.

4 The proposed integrated architecture and algorithms

4.1 A mixed associative model of decision-tree and chronicles

We have shown in the previous section that decision trees are weak in accounting for temporal evolutions, while a classical chronicle recognition approach does not handle correctly the context: at each point in time, an incoming event will only help discriminating between the current chronicle candidates, and hence the recognition process will behave in a *monotonic* narrowing way. But this complete set of chronicles can be huge, whereas at each point in time only a small subset is relevant. So we should find a way to directly relate the chronicles to the context in which they are applicable. The solution we propose is to mix decision-trees and chronicles in a unified mode: we simply replace actions in leaves by sets of chronicles, which are the possible ones in this special context. Anyway, there are still leaves consisting of simple actions, for either (1) in some contexts there are "obvious" reactions that do not depend on next evolutions of the world, or (2) there is a "hole" in the expertise knowledge, and no known chronicle match the context, then a "default" action should be applied.

The notions of priority in decision-trees and goodness value in chronicles can be mixed very easily, since they account for the same concept of "preference". Priorities serve for discriminating between different applicable contexts, hence between different sets of chronicles, and goodness values between different chronicles in a set.

One should be aware that the condition activating the chronicle must remain true during its execution. This means we should tune the condition accordingly in the branches of the tree for each chronicle it applies to. For instance, the condition *beyond-visual-range=true* must hold in our example chronicles. Then if it becomes false as soon as the distance between the two aircraft is below 2 miles, and if the execution of a chronicle Q_i can last long enough to reduce the distance by 1 mile, then the condition that must be true at the beginning of the chronicle is *distance(a,b) > 3 miles*. As for priorities, this tuning of conditions can be done dynamically, since it may depend on parameters that will only be known at execution time (e.g. aircraft speed, type of opponent aircraft, etc). We have chosen not to develop further those aspects in this paper, focusing on temporal ones. That's why our example is simplified with this respect.

4.2 The overall architecture

At each time, one must have:

- The (static, built off-line) decision-tree $\mathcal{D}$ with conditions on branches, priorities, and sets of relevant chronicles together with goodness values on leaves.
- The (dynamic, updated on-line) context description $\mathcal{C}$ containing fluents true at the current time.
- The (dynamic, updated on-line) current active leaves, i.e. the set of active candidate chronicles $\mathcal{SQ}$, and/or simple actions $\mathcal{SA}$.
- The (dynamic, updated on-line) current automaton $\mathcal{A}$ encompassing the next possible steps.

The system reacts to the reception of an event $\mathbf{e}_r$ according to the following *non-monotonic* process, in which the automaton is first pruned in step 2, since time has moved forward, then reinitiated in step 4 with a new clock, before steps 5 prune it again according to chronicles that are not candidate anymore. Steps 6 and 7 manage to integrate the new candidate chronicles and/or simple actions into the automaton, and step 8 takes the "best" decision, developing a game search algorithm. The two procedures `Build-automat` and `Play-automat` will be

described in next subsection.

1. Update the context $\mathcal{C}$ (as in classical decision-tree [4]).
2. if $\mathbf{e}_r$ lies on one of the next transitions in $\mathcal{A}$,
then the target state of this transition becomes the current state q_0, and the concurrent transitions are deleted..
else $\mathbf{e}_r$ is an unpredicted event that forces us to reset the automaton to a simple state q_0.
3. Activate $\mathcal{D}$ to get the new sets $\mathcal{SQ}$ and/or $\mathcal{SA}$.
4. Reset a clock $\mathbf{x}$ when entering $\mathcal{A}$ in q_0.
5. For all transition τ already in $\mathcal{A}$:
- if the chronicle it comes from is no longer in $\mathcal{SQ}$,
then remove τ from $\mathcal{A}$.
6. For all new chronicle Q_i, `Build-automat`*($\mathcal{A}$, Q_i, $\mathbf{e}_r$).*
7. For all new simple action, add a transition τ with no guard, and the decision-tree priority as goodness value.
8. `Play-automat`*($\mathcal{A}$).*

This overall process is summarized in figure 5, which gives back the architecture of figure 3 augmented with chronicles and the game automaton.

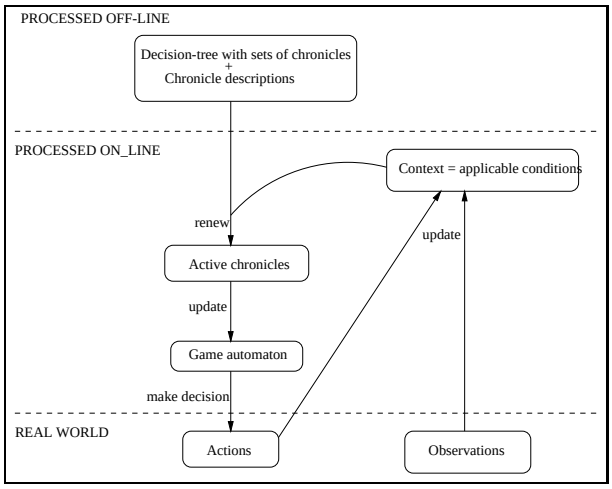

Figure 5: The new proposed architecture

4.3 The on-line process: decision making as a game procedure

We first give hereafter the sketch of algorithm for building new transitions in the automaton from a new chronicle candidate and the last received event, i.e. `Build-automat`($\mathcal{A}$, Q_i, $\mathbf{e}_r$):

For all $\mathbf{e}_i$ successor of $\mathbf{e}_r$ in Q_i:
if there is no transition with this label in $\mathcal{A}$, then
- add a new state q_{next} in $\mathcal{A}$.
- add a transition τ from q_0 to q_{next}, labeled by $\mathbf{e}_{next}$ and which type (activated or received) depends on the type of the event (action or observation).
- the interval values in the chronicle directly provides the guard on τ.
- transfer on q_{next} the goodness value lying in the corresponding chronicle.
else
- recompute the guard intersecting it with the interval constraint in Q_i.
- recompute the goodness value as the min of the current one and the goodness value of Q_i.

Figure 6 shows what results from applying it to our example. Receiving the event $\mathbf{b}_{mr}$ puts the pilot into a situation with the five new chronicle candidates of figure 4. Then we look for all the direct successors $\mathbf{e}_{next}$ of the last received event $\mathbf{b}_{mr}$, and get four of them. For each one a new state is added and the transition to this state, labeled with $\mathbf{e}_{next}$, is added as well, computing guards according to the corresponding chronicles. The goodness values are transferred as well, with here the case where an event ($\mathbf{a}_{acc}$) belongs to two different chronicles, and hence the min of the goodness values is computed.

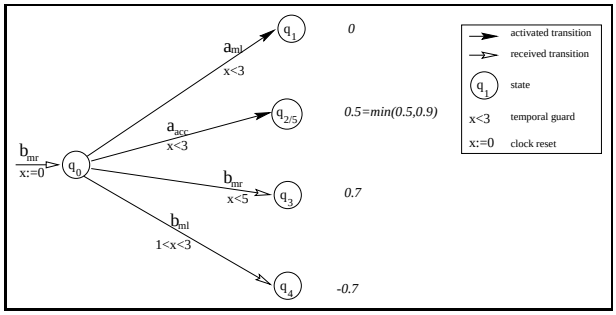

Figure 6: The first step controller

The procedure `Play-automat`($\mathcal{A}$) that will now identify the best decision to make through the automaton has been inspired by classical *controller synthesis* algorithms [2], but highly simplified in our case to single-depth transitions, and adapted to take into account in a classical game-like manner the goodness values on states. We will see that non-trivial temporal decisions can be taken, such as "wait some time and then do that". The step 2 is the key point of the algorithm given hereafter. It means checking if a transition τ can be compelled to occur before any other one, and it depends on the type of τ: if it is a received one, we simply check if the upper value of its guard is lower than the lower value of the guard of any other received transition. If it is an activated one, we will try to restrict the upper value of its guard to satisfy the same requirement (see [2] for details).

Reaching step 4 means we have a "safe" transition that should be selected. If it is a received one, we simply have to wait. If it is an activated one, we can try to optimize the decision, taking now into account (step 4.1) all the known better transitions that could not be enforced[1] : since any value of the guard of the selected transition can be taken safely, why not waiting while at least one of those better transitions is still possible ? In that case the decision consists in an action (labeling the selected transition) and a delay to wait before activating it.

0. BETTER:=$\emptyset$.
1. Select the best next state q_B *(highest goodness value).*
2. Check if the transition τ *to* q_B *can be enforced.*
3. If NOT then "forget it", i.e. push q_B *into the set* BETTER*, and GOTO 1.*
4. if $\tau \in T_b$ *then* DECISION:=wait.
else
4.1. for all $q_T \in$ BETTER*, reached by transition* τ'*,*
- if the guards $g(\tau)$ *and* $g(\tau')$ *intersect then*
lower$(g(\tau))$:=*min(*upper$(g(\tau))$,upper$(g(\tau')))$
4.2. DECISION:= ($\mathbf{a}_b$, *lower*$(g(\tau))$)
5. send DECISION *to the execution manager.*

This algorithm can be illustrated through the example of figure 6. In the first iteration, we get q_3 as the best transition (maximal goodness value 0.7), but then trying to enforce this transition fails: the transition to q_4 may uncontrollably occur before. Hence we "forget" q_3 and look for another solution, getting state $q_{2/5}$. Enforcing it with respect to remaining transitions succeeds and leads to restricting the guard to $\mathbf{x} < 1$. "Reminding" q_3 shows that actually waiting up to 1 second might issue a better state. Hence the final decision is "wait no more than 1 second and if nothing has happened accelerate". Then, if we suppose that nothing happens and the pilot accelerates, following the global loop algorithm, we get to a a new stage in which a new automaton is computed, which figure 7 illustrates. From the current state there are now three possible events. Since two of them belong to the same chronicle (namely Q_2) and should both happen, but in any order, we can even (though this is not mandatory and hence has not been included in the given algorithms) already compute a second level of transitions. Then guards are computed in the same way and the algorithm gives back that q_4 is the best but cannot be enforced, and any other transition is then equal. So the best is here simply to wait ...

[1]Notice that those "better" transitions are necessarily received ones: if they were activated, the algorithm would have preferably select them ...

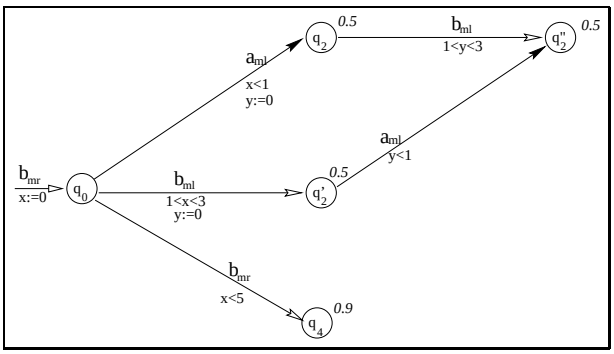

Figure 7: The second step controller

5 Discussion

The main strength of our approach is to use different models that are best suited for each of the different strong requirements of the application domain: the decision-tree for reasoning on the *context*, the chronicles for handling the *temporal* aspects, and the game automaton for processing an actual *predictive* decision-making process. Our architecture shows that it is rather straightforward to integrate those distinct models and make them work together in a smooth way.

Our approach can be compared to the work in [11], mainly based on task hierarchies (as in static plan recognition approaches like [6]), which considers the opponent actions while making dynamic decisions. In their case the same mechanism employed by the agent in generating its own behaviors is used for tracking others' behaviors. This can be quite computationally demanding and amounts to a single interpretation of the on-going actions. Our approach is more similar to the one used in a military context where the pilot is taught how certain manoeuvres of the opponent should be interpreted.

Another interesting comparison is with the TIGER project [8], where three models are mixed in a similar way: decision trees are replaced by compiled rule bases, and then a classical chronicle recognition mechanism feeds a model-based diagnosis system. In our case, we do more than mere supervision, having at the end of the chain a highly reactive decision-making system processing hypothetical reasoning on the future instead of comparative reasoning on the past, as in diagnosis. This compelled us to improve the chronicle structure to incorporate actions in it.

Concerning the game-based decision strategy, we can cite a work in similar military applications [7], where the decision is also dynamically made, but thanks to a discrete game approach. Moreover, classical heuristic techniques are used, exploring down the

tree in some kind of lookahead simulation, which in our case is not necessary since we directly inherit goodness values from our associative chronicle model. Hence we gain higher efficiency as well as higher reliability on the values, if compared to the min-max values in a discrete tree.

Talking about efficiency, it is easy to argue that it gets very high in our architecture, permitting to meet real-time requirements, thanks to the use of associative models (decision-trees and chronicles). The very new part (game automata building and search) is also very efficient since the automaton will always be kept small, both in *breadth* (the context handling restricts the number of chronicles, and therefore the number of alternative states, to consider) and in *depth* (one only needs to consider the next step, since we get goodness values from the corresponding chronicle).

To end with a less optimistic note, we should tell a few words about the off-line construction of the decision-tree with sets of chronicles, which is an expertise acquisition process that one has always to consider when dealing with associative models: *uncompleteness* and *inaccuracy* of such an expertise is always to be feared and is the topic of many on-going research works that are beyond the scope of this paper. Anyway, this problem is partly solved in our architecture since it can be made robust to unexpected events as well, thanks to the already mentioned "simple action" branches.

6 Conclusion

We have presented in this paper an original global architecture, mixing and adapting (both at the representation and reasoning levels) existing theoretical models in a specific application area, which we hope might contribute in bridging the gap between theoreticians and practitioners.

Our approach is relevant in cases where dynamic supervision and real-time decision-making interact in a highly reactive way, and complex temporal constraints must be accounted for. We strongly believe that our approach mixing symbolic reasoning models from the artificial intelligence community and control models from the theoretical computer science community can help making interesting advances in this field. As far as we know, such attempt has only been made in a work [3] that uses a similar kind of automata as simulation models of telecommunication networks, from which they compute a global chronicle model for efficient on-line supervision purpose. What is interesting is that we use automata downstream instead of upstream, hence in a complete different way, namely for game-like decision making, which keeps the efficiency of associative models.

Acknowledgments

Thierry Vidal wishes to thank the Excellence Center for Computer Science and Systems Engineering (ECSEL) in Linköping which supported him during this research work. 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

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

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

[3] S.Bibas, P.Dague, F.Lévy, M.O.Cordier & L.Rozé - *Scenario generation for telecommunication network supervision*, In Proc. of the IJCAI-95 Workshop on A.I. in Distributed Intelligent Networks, Montréal (Canada), 1995.

[4] S.Coradeschi, L.Karlsson & A.Törne - *Intelligent agents for aircraft combat simulation*, In Proc. of the 6th Conf. on Computer Generated Forces and Behavioral Representation, Orlando (FL, USA), 1996.

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

[6] J.Hong - *Plan recognition on the basis of structural and temporal constraints*, Current trends in AI planning, ed. by C.Backström and E.Sandewall, IOS Press, 1994.

[7] A.Katz & B.Butler - *"Game Commander"-Applying an architecture of game theory and tree lookahead to the command and control process*, In Proc. of the 5th Annual Conf. on AI, Simulation, and Planning in High Autonomy Systems, Gainesville (Fl, USA), 1994.

[8] A.Milne & al. - *TIGER: real-time situation assessment of dynamic systems*, Intelligent Systems Engineering, 103–124, Autumn 1994.

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

[10] E.Schwalb & R.Dechter - *Processing Temporal Constraint Networks*, technical report, Information and Computer Science, University of California, Irvine, January 1995 (to appear in Artificial Intelligence, summer 1997).

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

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