A unified dynamic approach for dealing with Temporal Uncertainty and Conditional Planning

Thierry Vidal

LGP/ENIT
47, av d'Azereix - BP 1629
F-65016 Tarbes cedex - FRANCE
thierry@enit.fr

Abstract

In temporal planning, Temporal Constraint Networks allow to check the temporal consistency of a plan, but it has to be extended to deal with tasks which effective duration is uncertain and will only be observed during execution. The Contingent TCN models it, in which Dynamic controllability has to be checked, i.e.: during execution, will the system be able to consistently release tasks according to the observed durations of already completed tasks ? This behaviour is a reactive one suggesting the plan is conditional in some sense. A Timed Game Automaton model has been specifically designed to check the Dynamic controllability. This paper furthermore discusses the use of such a model with respect to conditional and reactive planning, and its strength with respect to execution supervision needs, and suggests improving efficiency by partitioning the plan into subparts, introducing so-called waypoints with fixed time of occurrence. Last we show that the expressive power of automata might allow to address more elaborate reactive planning features, such as preprocessed subplans, information gathering, or synchronization constraints.

Background and overview

Temporal Constraint Networks (TCN) (Schwalb & Dechter 1997) rely on *qualitative* (or symbolic) constraint algebras (Vilain, Kautz, & van Beek 1989) but more specifically tackle *quantitative* (or numerical) constraints. They are now at the heart of many application domains, especially scheduling (Dubois, Fargier, & Prade 1993) and planning (Morris, Muscettola, & Tsamardinos 1998; Fargier *et al.* 1998), where a TCN might allow one to *incrementally* (i.e. at each addition of a new constraint) check the temporal consistency of the plan.

In realistic applications the inherent uncertain nature of some task durations must be accounted for, distinguishing between *contingent* constraints (whose effective duration is only observed at execution time, e.g. the duration of a task) and *controllable* ones (which instanciation is controlled by the agent, e.g. a delay between starting times of tasks): the problem becomes a decision-making process under uncertainty, and consistency must be redefined in terms of ***controllabilities***, especially the ***Dynamic controllability*** that encompasses the reactive nature of the solution building process in dynamic domains: this property allows one to check if it is feasible to build a solution in the process of time, each assignment depending only on the situation observed so far, and still needing to account for all the future unknown observations. In (Vidal & Fargier 1999) partial results about complexity and tractable subclasses were given and a first ad hoc algorithm was provided, based on a discretization of time. This work has been recently completed by the introduction of the ***Waypoint controllability*** feature (Morris & Muscettola 1999) (i.e. there are some time-points which can be assigned the same time of occurrence in all solutions). Adding wait periods in a plan allows to get this property checked, and then the problem may lie in a subclass in which Waypoint and Dynamic controllabilities are equivalent.

This paper goes further: relying on a Dynamic controllability checking method through an equivalent Timed Game Automaton, we will show that a small set of waypoints might also be used to partition the plan into subparts in which small size automata can be built to check Dynamic controllability only locally, hence providing a nice tradeoff between expressiveness, optimality of the plan and efficiency. The other contribution of this paper is to situate the automaton approach within the conditional and reactive planning area.

The paper is organized as follows. Section 2 provides the basic constraint network model and the Dynamic and Waypoint controllability definitions, and Section 3 gives the complete automaton-based Dynamic controllability checking method (those two sections being developped in more details in (Vidal 2000)). Then section 4 gives evidence of the pros and cons of the approach in planning, with respect to efficiency, reactive behaviours in presence of temporal uncertainty and execution supervision needs, before proposing the combined framework of partitioned Dynamic controllability. Section 5 eventually describes how the full expressiveness of timed automata may be used to address more complex features such as preprocessed subplans, information gathering, or synchronization constraints.

Contingent TCN and Controllability

In temporal planning, the planning process produces a Temporal Constraint Network to represent the temporal informations captured by the plan. This model relies on a reified logic framework (Vila & Reichgelt 1993) that separates the atemporal logical propositions from their temporal qualification, which are then interpreted in a temporal algebra framework (in our case a time-point based one), for which one uses a graph-based model on which we will focus in this section. This model is used for checking the temporal consistency of the plan.

We will barely recall here the Contingent TCN model and focus on the Dynamic controllability property, both being already described in (Vidal & Fargier 1999), and we will recall as well the Waypoint controllability property introduced in (Morris & Muscettola 1999). We will use the extended expressiveness and the unified characterization that appear in more details in (Vidal 2000).

We first recall the basics of TCNs (Schwalb & Dechter 1997). At the qualitative level, we rely on the time-point continuous algebra (Vilain, Kautz, & van Beek 1989), where time-points are related by a number of relations, that can be be represented through a graph where nodes are time-points and edges correspond to precedence ($\preceq$) relations. We can use the same time-point graph to represent quantitative constraints as well, thanks to the TCN formalism (Schwalb & Dechter 1997). Here continuous binary constraints define the possible durations between two time-points by means of temporal intervals. A basic constraint between x and y is $l_{xy} \leq (y - x) \leq u_{xy}$ equally expressed as $c_{xy} = [l_{xy}, u_{xy}]$ in the TCN. TCN a priori allow disjunctions of intervals, but we will restrict ourselves to the so-called *STP* (Simple Temporal Problem) where disjunctions are not permitted. A TCN is said to be consistent if one can CHOOSE for each time-point a value such that all the constraints are satisfied, the resulting instanciation being a ***solution*** of the STP modelled by the TCN. Then consistency checking of such a restricted TCN can be made through complete and polynomial-time propagation algorithms.

The TCN suits well the cases in which effective dates of time-points and effective durations of constraints are always chosen by the agent. If not, the problem has to be redefined in the following way.

A typology of temporal constraints

One needs first distinguishing between two different kinds of time-points: the time of occurrence of an *activated* time-point can be freely assigned by the agent, while *received* time-points are those which effective time of occurrence is out of control and can only be observed.

This raises a corresponding distinction between so-called ***controllable*** and ***contingent*** constraints (*Clb* and *Ctg* for short): the former can be restricted or instanciated ***by the agent*** while values for the latter will be provided (within allowed bounds) ***by the external world*** (see (Vidal & Fargier 1999) for details).

For instance, in planning, a task which duration is uncertain and will only be known when the task is completed at execution time will be modelled by a Ctg between the beginning time-point which is an activated one and the ending one which is a received one. As far as Clbs are concerned, we need to further distinguish between two cases.

- An *EndClb* between x and y is a Clb in the usual way we designed them until now: after x has occurred, the system can let time fly before assigning a value to the EndClb. If y is an activated time-point, then the EndClb is called a Free, and the agent can let time fly up to the upper bound of the constraint u_{xy} before deciding when to release y. If y is a received time-point, then it means the agent can freely restrict the duration interval (hence it is not a Ctg), but the effective value will be assigned only when y is received. This kind of EndClb is called a Wait. For instance, a delay between the end of a task and the beginning of the next one is usally a Free, while a constraint restricting the possible delay between the ends of two tasks (and which can be further restricted if needed) is a Wait.
- A *BeginClb* between x and y is a Clb which effective duration must be decided as soon as x has occurred. For instance, a task corresponding to a robot move may have different controllable durations depending on the robot speed that can be fixed by the agent, but this has to be done at the beginning of the task and will not be changed throughout the move. Hence the duration is determined when releasing the task.

The resulting Contingent Temporal Constraint Network model

The previous subsection introduces some level of uncertainty in the STP that results in the following definition of the corresponding *Contingent TCN*.

Definition 1 (CTCN)
$\mathcal{N} = (V_b, V_e, R_g, R_c)$ is a Contingent Temporal Constraint Network with

$V_b = \{b_1, \dots, b_B\} \cup \{b_0\}$*: set of the* B *activated time-points plus the origin of time,*

$V_e = \{e_1, \dots, e_E\}$*: set of the* E *received time-points,*

$R_c = \{c_1, \dots, c_C\}$*: set of the* C *Clbs, with* $R_c = R_{bc} \cup R_{ec}$*,* R_{bc} *being the set of EndClb and* R_{bc} *being the set of BeginClb,*

$R_g = \{g_1, \dots, g_G\}$*: set of the* G *Ctgs.*

The reader should take note that the distinction between the Free and the Wait is purely semantic, the forthcoming model, property definitions and reasoning processes will barely differentiate between Ctg, EndClb and BeginClb.

In the following, a ***decision*** $\delta(b_i)$ will refer to the effective time of occurrence of an activated time-point b_i, and an ***observation*** ω_i will refer to the effective duration of the Ctg between x_i and e_i (known by the system when receiving time-point e_i).

Dynamic and Waypoint controllabilities

In a CTCN, the classical consistency property is of no use. It would mean indeed that there exists at least one complete assignment of times of occurrence to the whole set of time-points such that each effective duration belongs to the corresponding interval constraint. But this would require that in such a "solution" a value is chosen for the Ctgs, which contradicts the inherent unpredictable nature of those constraints. The decision variables of our problem are only the activated time-points. And hence a solution should be here, intuitively, an assignment of the mere activated time-points such that all the Clbs are satisfied, whatever values are taken by the Ctgs. This suggests the definition of the so-called *controllability* property.

In (Vidal & Fargier 1999), three different levels of controllability have been exhibited, completed in (Morris & Muscettola 1999) by an additional property called the Waypoint Controllability. A unified definition of those four properties is proposed in (Vidal 2000), but we will focus here barely on the Dynamic and the Waypoint controllabilities that are of interest in planning.

Definition 2 (Situations)
Given that $\forall i=1\ldots G$, $g_i=[l_i,u_i]$,

$\Omega = [l_1,u_1] \times \ldots \times [l_G,u_G]$ *is called the* space of situations, *and*

$\omega = \{\omega_1 \in [l_1,u_1],\ldots,\omega_G \in [l_G,u_G]\} \in \Omega$ *is called a* situation *of the CTCN.*

Then, for each time t, one can define the current-situation $\omega_{\preceq t} \subseteq \omega$ *which is the set of observations prior to t, i.e. such that only Ctgs with ending points* $e_i \preceq t$ *are considered.*

In other words, a *situation* represents one possible assignment of the whole set of Ctgs, and a *current-situation* with respect to t is a possible set of observations up to time t.

Definition 3 (Schedules)
$\delta = \{\delta(b_1),\ldots,\delta(b_B)\} \in \Delta$ *is called a* schedule, Δ *being the* space of schedules *(i.e. the cartesian product of interval constraints* $(b_i - b_0)$*)*

Then, for each time t, one can define the current-schedule $\delta_{\preceq t} \subseteq \delta$ *which is the sub-schedule assigned so far, s.t.* $\forall x_i \in V_b \cup V_e$ *with* $x_i \preceq t$,
 if $\exists c_{ij} = (b_j - x_i) \in R_{bc}$ *then* $\delta(b_j) \in \delta_{\preceq t}$
 else if $x_i \in V_b$ *then* $\delta(x_i) \in \delta_{\preceq t}$

A *schedule* is then one possible sequence of decisions (that might be "controllable" or not). And a *current-schedule* encompasses the notion of reactive chronological building of a solution in plan execution. One should notice that we take into account the case of BeginClb, for which the time of occurrence of the ending point b_j might have been already decided at time t (and hence must be in $\delta_{\preceq t}$) even if $t \preceq b_j$.

Definition 4 (Projection and Mapping)
$\mathcal{N}_\omega$ *is the* projection *of* $\mathcal{N}$ *in the situation* ω*, built by replacing each Ctg* g_i *by the corresponding value* $\{\omega_i\} \in \omega$*. In (Vidal & Fargier 1999) a projection is proved to be a simple TCN corresponding to a STP.*

μ *is a mapping from* Ω *to* Δ *such that* $\mu(\omega) = \delta$ *is a schedule applied in situation* ω.

Intuitively, a CTCN will be "controllable" if and only if there exists a mapping μ such that every schedule $\mu(\omega)$ is a solution of the projection $\mathcal{N}_\omega$. In fact this is only the "weakest" view of the problem (called Weak controllability in (Vidal & Fargier 1999)), that does not take into account the relative temporal placement of decisions and observations. The above intuitive statement implicitely assumes that one knows the complete situation before choosing one schedule that will fit. But when a plan is executed, decisions are taken in a chronological way, and for each atomic decision the agent knows the past observations, but the observations to come are still unknown. The Dynamic controllability property defined below allows to take this into account, being more restrictive than the Weak controllability, in that it compels any current schedule at any time t to depend only upon the current situation at that time.

Another interesting property that we will use in this paper, called Waypoint controllability has been further introduced in (Morris & Muscettola 1999), stating that there are some points for which all the schedules share the same time of occurrence, whatever the situation is. Those waypoints serve as "meeting" time-points in a plan, when the agent waits for all the components of a subpart of the plan to be over before starting the next stage. Waypoints can be created during the planning process through the addition of "wait periods" (Morris & Muscettola 1999).

These definitions are given herebelow, where one can notice that they both share the first condition (which is the "weak" part), and have to satisfy a second one that fits the corresponding description above, but has a similar formulation in both cases: this unified definition will make it easy to combine the two properties in the global framework that will be presented in this paper.

Definition 5 (Dynamic/Waypoint controllability)
- $\mathcal{N}$ *is* Dynamically controllable *iff*
  (1) $\exists\mu : \Omega \longrightarrow \Delta$ *s.t.* $\mu(\omega)=\delta$ *is a solution of* $\mathcal{N}_\omega$
  (2) $\forall(\omega,\omega') \in \Omega^2$*, with* $\delta=\mu(\omega)$ *and* $\delta'=\mu(\omega')$*,*
  $\forall t$*, if* $\omega_{\preceq t} = \omega'_{\preceq t}$ *then* $\delta_{\preceq t} = \delta'_{\preceq t}$
- $\mathcal{N}$ *is* Waypoint controllable *with respect to* $W \subset V_b$ *iff*
  (1) $\exists\mu : \Omega \longrightarrow \Delta$ *s.t.* $\mu(\omega)=\delta$ *is a solution of* $\mathcal{N}_\omega$
  (2) $\forall(\omega,\omega') \in \Omega^2$*, with* $\delta=\mu(\omega)$ *and* $\delta'=\mu(\omega')$*,*
  $\forall x \in W$*,* $\delta(x) = \delta'(x)$

Those definitions are illustrated in (Vidal 2000) through small examples that will not be given here.

Dynamic and Waypoint controllability are proven to be exponential in the number of time-points. But the complexity of Waypoint controllability is actually exponential in the maximum number of time-points between two waypoints (Morris & Muscettola 1999), which means the more waypoints one considers in the CTCN, the less complex Waypoint controllability is.

We will now focus on Dynamic controllability issues. The checking algorithm given in (Vidal & Fargier 1999) is a "home made" one that relies on a discretization of time. Considering the inherent continuous nature of interval constraints in TCNs, we have been interested in the applicability of timed automata in this context. The next section describes a method based on such a model, which appears to be perfectly suited for solving our problem.

Using a Timed Automaton for Dynamic Controllability checking

The principles of timed automata

We will assume in the following the reader has a minimal knowledge of finite-state automata. The method we present here relies on the *timed automata* model used for describing the dynamical behaviour of a system (Alur & Dill 1994). It consists in equiping a finite-state automaton with time, allowing to consider cases in which the system can remain in a state during a time T before making the next transition. This is made possible by augmenting states and transitions with "continuous variables called *clocks* which grow uniformly when the automaton is in some state. The clocks interact with the transitions by participating in pre-conditions (*guards*) for certain transitions and they are possibly reset when some transitions are taken." The original model can be slightly changed converting some guards into *staying conditions* (Asarin, Maler, & Pnueli 1995) in states (see (Vidal 2000) for more details). The expressiveness remains the same, but those are more convenient to use when addressing controllability checking.

In our model (see next subsection), we will introduce a second type of clocks, that work in the opposite way: they are first set to a fixed value, then they decrease uniformly until they reach the value 0, which serves as a specific guard for activating some transitions. To distinguish them, we have chosen to call the first type of clocks *stopwatches* and the second one *egg timers* ... We will see that *stopwatches* allow to model Ctgs and EndClbs, while *egg timers* appear to be necessary to model BeginClbs.

In (Maler, Pnueli, & Sifakis 1995; Asarin, Maler, & Pnueli 1995), such tools are claimed to be well-suited to *real-time games*, where transitions are divided in two groups (such as constraints in CTCN) 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 controlled transitions that will lead her to one of her winning states. This extension of the classical discrete game approach has the following features: (1) there are no "turns" and the adversary need not wait for the player's next move (Maler, Pnueli, & Sifakis 1995), and (2) each player not only choose between alternative transitions, but also between *waiting* or not before taking it, from which one can view the two-player game as now "a three-player one where Time can interfere in favor of both of the two players (Asarin, Maler, & Pnueli 1995)".

This is especially interesting for controlling reactive systems in which one player is the *environment* ("Nature") and the other player is the *controller* (for instance a plan execution controller) and has control only over some of the transitions (similarly as Clbs in CTCN). A *trajectory* (i.e. a path in the automaton) reaching a winning state for the controller is called *C-trajectory* in (Maler, Pnueli, & Sifakis 1995), and defines a so-called *safety game* (Asarin, Maler, & Pnueli 1995) policy.

An accurate Timed Automaton model

Different formulations of timed automata models can be found in the bibliography of the area. We propose here our own adapted definition of Timed Automaton, which relies on the preliminary definition of conditions and functions that can be applied to clocks, namely: Re is the finite set of all possible stopwatch reset functions (of the form $(sw_i \leftarrow 0)$ s.t. $sw_i \in \Gamma_{sw}$), As is the infinite set of all possible egg timer assignment functions (of the form $(et_i \leftarrow \Delta_i)$ s.t. $et_i \in \Gamma_{et}$ and $\Delta_i \in \mathbb{Z}$) and Cond defines the infinite set of all possible clock conditions that will be used as guards or staying conditions (generally of the form $(l_i \leq clock_i \leq u_i)$ s.t. $clock_i \in \Gamma$ and $(l_i, u_i) \in \mathbb{Z}^2$ or $(et_i = 0)$ s.t. $et_i \in \Gamma_{et}$). Details on those sets can be found in (Vidal 2000). One should just notice here that Cond defines both ranges of values that must/will be reached by a stopwatch and conditions expressing that an egg timer has reached the value 0.

Definition 6 (Timed Game Automaton) *$\mathcal{A} = (Q, \Sigma, \Gamma, S, T)$ is a timed game automaton (TGA) where*

- *Q is the discrete and finite set of states q_i, with three special cases:*
  - *q_0 is the initial state,*
  - *q_{ok} is the unique* winning *state,*
  - *$q_\dagger$ is the unique* losing *state;*
- *$\Sigma = \Sigma_b \times \Sigma_e$ is the input alphabet such that any label in Σ_b is of the form b_i and any label in Σ_e is of the form e_i;*
- *$\Gamma = \Gamma_{sw} \cup \Gamma_{et}$ is the discrete and finite set of clocks, where Γ_{sw} is the set of* stopwatches *and Γ_{et} is the set of* egg timers*;*
- *$S : Q \rightarrow$ Cond assigns staying conditions to states;*
- *$T = T_b \cup T_e \subseteq Q^2 \times \Sigma \times$ Cond $\times$ Re $\times$ As is the set of transitions of the form*
  $\tau = <q, q', \sigma, g, r, a>$ with a distinction between
  - *$\tau \in T_b$ is an* activated *transition* iff *$\sigma \in \Sigma_b$,*
  - *$\tau \in T_e$ is a* received *transition* iff *$\sigma \in \Sigma_e$.*

Therefore for activated transitions, if there is a guard conditionning its activation, the agent will be able to decide the exact time of activation by "striking" the stopwatch within the two bounds of the guard. If it is a received transitions, then the transition will be automatically taken at some unpredictable time within the two bounds of the guard. Egg timers will on the contrary compel some activated transitions to be aken at one and only one predicted time.

A first remark can be made about the losing state. Obviously an agent will never make a move that will push herself into the losing state, which means that

$$\text{if } \tau = <q, q_\dagger, \sigma, g, r, a>, \text{ then } \tau \in \mathrm{T}_e$$

Building and using a TGA for Dynamic controllability checking

In (Vidal 2000), we thoroughly demonstrate that a TGA can be built out of the CTCN and is able to express exactly what a CTCN expresses (actually more, see the last section). We will just exhibit here some intuitive correspondences between the two models. Obviously letters b and e and terms *activated* and *received* refer to the same concept in both models: an event in $\mathcal{N}$ will appear as a translation labelled with this event in $\mathcal{A}$. Therefore $\Sigma_b \equiv V_b$ and $\Sigma_e \equiv V_e$. Similarly, temporal intervals in $\mathcal{N}$ will appear as guards in $\mathcal{A}$.

Then (Vidal 2000) provides an algorithm acting on the TGA that is proven to check Dynamic controllability. This method is inspired by well-established techniques in the area of reactive program synthesis. A *synthesis* algorithm uses as a basic principle a so-called *controllable predecessors* operator: unformally, this operator computes from a state q the states from where the system can be forced to reach q, "returning revised guards and staying conditions (Asarin, Maler, & Pnueli 1995)". Hence a synthesis algorithm will recursively apply this operator from the initial set of winning states until it reaches a fixed point. If q_0 is in the final set, then the controller can always win the game. Moreover, the synthesis process is designed to work in an incremental way: it is straightforward to apply the operator only to the set of newly added states.

On the application of TGA in planning

Relevance and conditional nature of the approach

First, we don't really need to argue on the relevance of representing temporal uncertainty in planning, since durations of tasks are often not fully predictable in realistic applications. Talking about uncertainty, our approach does not address unpredicted events: we only consider events that are actually known to occur, it is only their time of occurrence that is unknown (though it is known to lie within a temporal window). This kind of expressiveness is needed in most of the current planning and scheduling applications, as for instance in the NASA project Deep Space One (Morris, Muscettola, & Tsamardinos 1998; Morris & Muscettola 1999), or in multimedia authoring tools (Fargier *et al.* 1998), a multimedia document (with audio and video units) being actually a partial-order plan being executed during a browsing session.

Interestingly enough, having to deal with such contingent durations leads an agent to base its activation decisions on previous observations. Here again it is only the time of activation that is considered as a decision variable. For instance, consider a task that must be activated at b_2 within 5 seconds before or after e_1 occurs. Figure 1 shows the TGA that results from a synthesis algorithm: one shouldn't decide to activate b_2 less than 25 seconds after b_1 to make sure the constraints will be satisfied, but then it is possible that e_1 is received before, and in that case one just has to activate b_2 within 5 seconds. Therefore the example is Dynamically controllable. In this example we have in fact two different sequences of events depending on ω_1 being lower or greater than 25. The resulting plans are different, which means this example corresponds to some kind of conditional plan. Moreover, the TGA models a reactive (or game-like, as the name suggests it) behaviour, since the decision initially made in q_1 to activate b_2 at 25 might be opportunistically modified on early reception of e_1. One might get b_2 actually activated only 10 seconds after b_1.

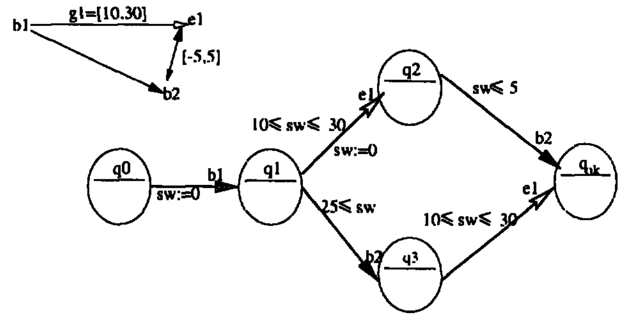

Figure 1: An example of reactive behaviour

This example might also illustrate the case of BeginClbs: suppose that $(b_2 - b_1)$ is a BeginClb, then it means the time of occurrence of b_2 must be set in q_1, which will be modelled thanks to an egg timer assigned in that state, and the transition to q_3 triggered as soon as the egg timer equals 0. Here there is no possibility to be reactive, and the synthesis algorithm would fail, concluding Dynamic controllability does not hold.

What lesson can be learned from this example ? The TGA is a discrete event tool that implicitely models reactive behaviours and is hence able to synthesize conditional plans. This is simply because two transitions from the same state correspond to a branching, i.e. a OR node in the model, which means only one of the two next states will be traversed. On the contrary, in a CTCN time-points are AND nodes: all the next time-points need to be effectively received or activated. Consequently, CTCNs are able to implicitely model some time-based branching features in a plan, but to check the temporal feasibility of the plan, one needs to develop those branches for instance in the TGA (or through a discrete game simulation as in (Vidal & Fargier 1999)).

As a matter of conclusion, the CTCN formalism is a very powerfull tool for describing the ***specifications*** of a dynamic system (as e.g. a contingent plan), through the constraints it has to meet, expressing rich temporal information in a compact way. Whereas the TGA is a ***simulation*** model that captures all the possible ***execution scenarii*** of the plan: it has the advantage of providing efficient and robust techniques to check its "safety", but runs the risk of combinatorial explosion in the number of states.

The TGA as a plan execution control tool

Another strength of this approach is to provide a model that can be directly used as an execution supervisor tool. As a matter of fact, as far as timed automata are concened, a distinction can be made between the *Analysis* problem and the *Synthesis* one (Maler, Pnueli, & Sifakis 1995). Synthesis means building a *controller* automaton out of the original one, synthesizing its conditional behaviour in reaction to the environment. Analysis means proving that the system is *controllable*, i.e. proving that a controller can be built. This yields a natural parallel between Synthesis and Analysis in timed automata on one hand and respectively solution computation and satisfiability checking in constraint-based temporal planning on the other hand. Not only solving the synthesis problem clearly implies solving the analysis one, but it gives a schedule of starting times for the planned tasks. Moreover, in constraint networks only a deterministic sequence of decisions might be issued, whereas with a TGA one can get a reactive execution supervisor, with disjunctive possible trajectories, which makes it a much more powerfull tool in dynamic and uncertain planning environments.

Complexity and practical efficiency issues

As far as complexity is concerned, the algorithms for building and synthesizing the automaton are respectively linear and in logarithmic time in the number of states, which is in the worst case p^B, where B is the number of activated time-points and p the *degree of parallelism* (i.e. the maximum number of time-points possibly occurring at the same time) (Vidal 2000). Hence the complexity of the approach lies in the possibly exponential number of states developped, which depends upon the network feature p.

First, the method might be relevant in application domains in which p is kept rather low, which is often the case in planning and scheduling, where the set of tasks to trigger is mostly a sequence, the partial order in the CTCN only adding some flexibility. Next, one could use *dispatchable* networks (Morris, Muscettola, & Tsamardinos 1998), that are TCNs in which redundant constraints are removed and only minimal paths are exhibited, so as to optimize propagation during execution. This could restrict as well the number of states produced in the TGA. Besides, automata-based techniques might be improved to reduce the number of states produced, considering that two subsequences containing the same set of events, though in distinct orders, might converge on the same state, or using more complex abstraction views, like in the so-called ***symbolic approaches*** (like ***Decision Binary Diagrams***) (Maler, Pnueli, & Sifakis 1995; Asarin, Maler, & Pnueli 1995). Another interesting suggestion (Bornot, Sifakis, & Tripakis 1997) is to first translate the CTCN into a timed Petri net, which is a rather compact representation, then using a system like KRONOS (Yovine 1997) that generates the TGA with very high performance. That could be relevant in domains where one has to deal with concurrent systems, like in multi-agent planning for instance, for which Petri nets would be especially well-suited.

Another possibility is to accept an incomplete checking algorithm in the long term (based on incomplete propagations in the CTCN), using the TGA only in the short term, as far as execution runs, so as to account for safety: the algorithm anticipates the possible losing state deadends and can activate any necessary recovery action in advance.

Last, it is argued in (Morris & Muscettola 1999) that choosing cleverly the set of waypoints through addition of some "wait" periods in the plan might lead to Dynamic controllability being equivalent to Waypoint controllability. Anyway, trying to design a plan in this way may lead to a high number of waypoints lowering the plan optimality. We will show in next subsection how one can mix waypoints and TGA to get some nice compromise.

An extended framework using TGAs and waypoints

As discussed before, one can introduce wait periods in a plan, which end points will be waypoints. The more one adds waypoints, the less hard it will be to prove Waypoint controllability and the more chances one has to get Dynamic controllability equivalent to Waypoint controllability (since the equivalence property is based on received points being *protected* from one another by putting waypoints between some of them (Morris & Muscettola 1999)). But this may severely restrict the optimality of the plan, since in that case an opportunitic scenario like the one in Figure 1 would not be possible. In other words, one may compel the executed plan to "play for time" at some points when it is not really needed.

But waypoints might be used barely to restrict Dynamic controllability checking in all subparts of the networks between any pair of waypoints. For doing so, we need to prove the following property, where $\mathcal{N}_{(x,x')}$ stands for the restriction of $\mathcal{N}$ to the time-points temporally constrained to be after x and before x'.

Property 1 (Partitioned Dynamic controllability) *$\mathcal{N}$ is Dynamically controllable <u>iff</u>*
(1) $\mathcal{N}$ is Waypoint controllable wrt $W \subset V_b$, and
(2) $\forall (x,x') \in W^2$ s.t. $x \preceq x'$ and $\nexists x'' / x \preceq x'' \preceq x'$, $\mathcal{N}_{(x,x')}$ is Dynamically controllable

Sketch of proof. The proof is rather straightforward. Dynamic controllability means that a current-schedule will only depend on the corresponding current-situation. For any time t between two waypoints x and x', $\delta(x)$ is set in all schedules, which means it does not even depend on $\omega_{\preceq\delta(x)}$, and any forthcoming decision will not depend on it either. Consequently, $\delta_{\preceq t}$ does only depend on the part of the situation between $\delta(x)$ and t, which is equivalent to the formulation above. ◇

Then, a possible global algorithmic framework to check Dynamic controllability using waypoints could be the following (that could be easily defined in an incremental way):

1. Use some heuristic (to be defined at the planning engine level) to decide to introduce waypoints or not while planning
2. Check if the CTCN is Waypoint controllable
3. Check Dynamic controllability by building a TGA between any pair of successive waypoints

If the heuristics are well defined, then the added waypoints will correspond to wait periods, which are chosen such that it is possible to set their value in all schedules, which means the CTCN will by construction be necessarily Waypoint controllable, and the second step might be removed. The idea is to introduce "not so many" waypoints in the plan, so as to still meet in one hand high optimality requirements for the plan, while on the other hand drastically reducing time and space complexity of the TGA approach by only synthesizing automata corresponding to subparts of the whole plan. This decomposition technique hence offers a nice trade-off between expressiveness, optimality and efficiency.

Using the full expressiveness of TGAs in planning

We will end this paper with some general considerations about the expressive power of the TGA formalism, that is obviously larger than what we use in the case of CTCN Dynamic controllability checking, and might be of interest for other planning problems.

Generalized conditional planning

A first obvious remark is about the relation between the TGA model and conditional or reactive planning: if the TGA allows to represent the inherent conditional nature of a CTCN, then why not using it for different kinds of branching in planning ? For instance, consider information gathering problems, in which a perception action is included in a plan, and the next steps of the plan depend on the outcome of this action. Or more generally speaking, all cases in which non-determism of actions has to be dealt with. If one wishes to represent distinct evolutions of a plan, then this corresponds to some disjunctions (OR nodes) that are naturally represented in a TGA and may be smoothly merged with temporal contingency branching. Hence TGA might be used in such cases as well. The only difference is that this kind of non-determinism cannot be represented in a compact way in a CTCN, and one can hardly avoid the use of OR nodes in addition to partial order (i.e. AND nodes) in a temporal constraint-based planning graph.

Preprocessed plans and reactive planning

Sometimes a unique plan with branching nodes is not enough to solve a problem. One may need to use a library of subplans to run in reaction to typical events. For instance a planning system may produce a nominal plan together with a number of alternative "recovery" sequences to be runned in replacement when a modeled disturbance occurs, as in (Washington, Golden, & Bresina 1999). Instead of having those subsequences connected to a node of the nominal plan, they may be stored in a library and connected to a type of received and unpredictable event, which defines a more general kind of uncertainty than the one addressed in this paper (not only the time of occurrence of the event is unknown, but the occurrence of the event itself). Receiving this event will force the execution supervisor to temporally abandon the current plan to run the corresponding recovery sequence.

Then a more general reactive framework needs to be designed, as in (Vidal & Coradeschi 1999): one can define temporal *chronicles* corresponding to each "abnormal" scenario, with the possibility of having several "faulty" events in elaborated and rather complete scenarios. A purely reactive TGA framework can be designed, where on-line automaton building is processed, in reaction to received events: the system dynamically matches the received event with the chronicles that contain it, and synthesizes the possible next steps in those chronicles in one unique incremental automaton.

By mixing the general off-line planning framework presented in this paper with this purely reactive behaviour, one get a real-time planning system that might be very robust in stochastic environments.

Synchronization constraints

In multimedia documents (Fargier *et al.* 1998) for instance, one has to model temporal constraints that are outside the scope of classical temporal algebras, called synchonization constraints. Three of them have been defined:

Parmin Two objects A and B are related by a parmin if both starts at the same time and the first that is finished terminates the other as well (for instance receiving a button click will stop prematurely a video sequence, while the end of the video sequence will cause the button to disappear, irrespective of the initial possible durations of both objects);

Parmaster This is the same as parmin, except that only the first object in the relation forces the second one to finish at the same time;

Parmax Two objects A and B are related by a parmax if both starts at the same time and the first that is finished has to wait for the second one to finish as well before one can process next steps in the plan.

The two first ones are interruption-like behaviours, while the third one is an "appointment" constraint. In (Fargier *et al.* 1998), a first description of these constraints is given, and the shortcomings of CTCN for modeling them is shown: a parmin for instance could only be modeled in a CTCN by considering three different ending events: (1) the expected end of the first object, (2) the expected end of the second object, and (3) the "effective end" of both, that will actually be one of the two previous ones. Representing a parmin hence means considering a ternary constraint involving the three time-points, which is not covered by CTCN where only binary constraints are allowed.

Interestingly enough, those behaviours are implicitely conditional behaviours, and are very easy to model through a TGA. Therefore, once again we have exhibited a type of feature needed in some application domains, for which CTCN are not expressive enough, but that could easily be represented through a TGA.

Conclusion

This paper has brought to light the advantage of using Timed Game Automata for checking dynamic temporal properties of a plan in the presence of temporal uncertainties. Discussing efficiency and usefullness in practice, it suggests the addition of heuristically and sparingly selected wait periods in the plan to partition it so as to be able to check the Dynamic controllability property locally, processing small size automata, hence controlling the expected state combinatorial explosion.

A discussion has been initiated as well on the applicability of Timed Automata to more general conditional and reactive planning features. We hope this study will suggest further studies in that field, mixing some of the approaches suggested, thus contributing to bridge the gap between symbolic (specification) models and discrete event systems (simulation) models in order to address realistic real-time planning problems in dynamic and uncertain environments.

Acknowledgement

The author is grateful to Paul Morris (NASA Ames Research) for fruitful discussions and his suggestion of the modified Dynamic controllability definition.

References

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

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

Bornot, S.; Sifakis, J.; and Tripakis, S. 1997. Modeling urgency in timed systems. *COMPOS'97, LNCS.*

Dubois, D.; Fargier, H.; and Prade, H. 1993. The use of fuzzy constraints in job-shop scheduling. In *IJCAI-93 Workshop on Knowledge-Based Planning, Scheduling and Control.*

Fargier, H.; Jourdan, M.; Layaïda, N.; and Vidal, T. 1998. Using temporal constraint networks to manage temporal scenario of multimedia documents. In *ECAI-98 workshop on Spatio-Temporal Reasoning.*

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science.*

Morris, P., and Muscettola, N. 1999. Managing temporal uncertainty through waypoint controllability. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on A.I. (IJCAI-99)*, 1253–1258. Stockholm (Sweden): Morgan Kaufmann.

Morris, P.; Muscettola, N.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-98).*

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

Vidal, T., and Coradeschi, S. 1999. Highly reactive decison making: a game with time. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on A.I. (IJCAI-99)*, 1002–1007. Stockholm (Sweden): Morgan Kaufmann, San Francisco, CA.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11:23–45.

Vidal, T. 2000. Controllability characterization and checking in contingent temporal constraint networks. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000).* Breckenridge (Co, USA): Morgan Kaufmann, San Francisco, CA.

Vila, L., and Reichgelt, H. 1993. The token reification approach to temporal reasoning. Technical Report 1/93, Dept of Computer Science, UWI.

Vilain, M.; Kautz, H.; and van Beek, P. 1989. Constraint propagation algorithms: a revised report. In *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufman.

Washington, R.; Golden, K.; and Bresina, J. 1999. Plan execution, monitoring, and adaptation for planetary rovers. In *IJCAI-99 workshop on Scheduling and Planning meet Real-Time Monitoring in a Dynamic and Uncertain World*, 9–15.

Yovine, S. 1997. Kronos: a verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer* 1(1/2).