Planning Robust Temporal Plans A Comparison Between CBTP[*] and TGA[†] Approaches[‡]

Y. Abdeddaïm[‡‡], E. Asarin[††], M. Gallien[**], F. Ingrand[**], C. Lesire[**], M. Sighireanu[††]
[**] LAAS/CNRS, University of Toulouse, France
[††] LIAFA, University of Paris 7 and CNRS, France
[‡‡] ESIEE Paris, COSI Laboratory, France

Abstract

Planning for real world applications, with explicit temporal representation and a robust execution is a very challenging problem. To tackle it, the planning community has proposed a number of original and successful approaches. However, there are other paradigms "outside" the Automated Planning field which may prove to be successful with respect to this objective. This paper presents a comparison of two "planning" approaches dealing with temporal and/or discrete uncertainties and with a strong emphasis on robust execution. The first approach is based on chronicles and constraint satisfaction techniques; it relies on a causal link partial order temporal planner, in our case IxTeT. The second approach is based on timed game automata and reachability analysis, and uses the UPPAAL-TIGA system. The comparison is both qualitative (the kind of problems modeled and the properties of plans obtained) and quantitative (experimental results on a real example). To make this comparison possible, we propose a general scheme to translate a subset of IxTeT planning problems into UPPAAL-TIGA game-reachability problems. A direct consequence of this automated process would be the possibility to apply validation and verification techniques available in the timed automata community.

Introduction

Decisional autonomous embedded systems are, by their very nature, real-time programs. *Automated planning* is one of the decisional processes required on these systems. Although the corpus of research is large for untimed planning approaches, the results are still modest when it comes to properly handle time and resources. Nevertheless, some planning solutions dealing with time and resources have been proposed and demonstrated (Chien 2006; Frank & Jónsson 2003; Lemai & Ingrand 2004).

On the other hand, the *validation and verification* field proposes techniques for the synthesis of timed controllers (Maler, Pnueli, & Sifakis 1995; Asarin & Maler 1999; Abdeddaïm, Asarin, & Maler 2006; Tripakis & Altisen 1999). These techniques are based on (1) timed game automata models and (2) logics allowing to express the properties to be satisfied by these models. The decisional process is described in this framework using strategies. However, the application of these techniques to real-world problems has been possible only recently due to an efficient algorithm proposed in (Cassez *et al.* 2005) and implemented in UPPAAL-TIGA (Behrmann *et al.* 2007).

This paper's contribution can be resumed as follows. We consider two methods, CBTP and TGA, for specifying and solving realistic planning problems asking for continuous time, duration uncertainty, uncertain ordering of events, and executability of plans. For these methods we consider two implementations IxTeT and UPPAAL-TIGA. First, we develop a mapping from a subclass of IxTeT problems into TGA reachability problems. This allows to verify and validate the original plan model using classical model-checking techniques. Second, we compare the methods and their implementations (1) qualitatively on general domains and (2) quantitatively on a special domain inspired from a planetary exploration rover mission. The qualitative comparison takes into account a large spectrum of criteria: the kind of problems that can be modeled, the flexibility of plans and strategies produced, their size and their execution.

Related works consider the use of verification techniques to do planning (e.g., (Cimatti *et al.* 1997; Jensen & Veloso 2000)) or to validate planning domain models (e.g., (Lowry, Havelund, & Penix 1997; Khatib, Muscettola, & Havelund 2001)). Beyond these works, few consider timed planning problems. (Khatib, Muscettola, & Havelund 2001) provides a mapping from interval-based temporal relations models to timed automata models. However, their work aims only to validate plans and does not consider the game semantics needed to use of model-checking as planning engine. (Vidal 2000) translates Contingent TCN in TGA models in order to verify and execute them, but does not consider producing the plan, nor expressiveness issues.

Illustrative Example

Throughout the paper, we use an example domain (*Explore*) inspired from a Mars rover (Ai-Chang *et al.* 2003) exploring an initially unknown environment. The rover can: move with the cameras pointing forward; move the cameras (mounted on a pan&tilt unit); take pictures (while still) with the cameras pointing downward, and communicate (while still). The mission of the rover is to navigate to take pictures of predefined locations, and to communicate with an orbiter

[*]Constraints Based Temporal Planning

[†]Timed Game Automata

[‡]This work has been supported by the National French Agency for Research (ANR) project AMAES. Part of this work has been funded by a grant from the European Social Fund (ESF).

during predefined visibility windows. There is a lot of temporal uncertainties, especially in the duration of navigations and communications.

Specifying Planning Problems

IxTeT Domain Description Formalism

Let $(\mathcal{P}_i, \mathcal{T})$ be a planning problem with $\mathcal{P}_i$ the initial plan and $\mathcal{T}$ the set of possible tasks $\{\tau_1, \tau_2, \ldots, \tau_p\}$. The tasks and the initial plan are augmented chronicles.

A *chronicle* (Ghallab, Nau, & Traverso 2004) for a set of state variables $a_1(), a_2(), ..., a_n()$ is a pair $\Phi = (F, C)$, where F is a set of temporal assertions about the state variables $a_1(), a_2(), ..., a_n()$ and C is a set of constraints on variables used in the chronicle.

A *state variable* $a(p_1, \ldots, p_k)$ is a piece-wise constant function $f_a(t, p_1, \ldots, p_k)$ with $t \in \mathcal{R}^+$ the time and $p_1, \ldots, p_k$ the possibly empty set of parameters. A state variable is defined for each value of t and may be undefined for some tuples $< p_1, \ldots, p_k >$. The evolution of *state variables* over the planning horizon is described using temporal assertions. A *temporal assertion* on a state variable $a(p_1, \ldots, p_k)$ is either an *event* or a *persistence condition*. They refer to decision instants (called *time points*) which are represented by *temporal variables* (with values in $\mathcal{R}^+$).

- An *event*, denoted *event*$(a(p_1, \ldots, p_k) : (?v_1, ?v_2), t)$, specifying an instantaneous change of the value of $a(p_1, \ldots, p_k)$ from value $?v_1$ to value $?v_2$ $(?v_1 \neq ?v_2)$ at time point represented by variable t, or
- A *persistence condition*, denoted *hold*$(a(p_1, \ldots, p_k) : ?v, (t_1, t_2))$, specifying that the value of $a(p_1, \ldots, p_k)$ must be equal to $?v$ during the interval $]t_1, t_2[$.

A *plan* $\mathcal{P}(S, \Phi, G, \text{CA}, T)$ is described by a finite set of state variables S, Φ a chronicle, CA $\subset \Phi$ a chronicle of uncontrollable events on state variables in S, $G \subset \Phi$ the set of goals expressed using persistence conditions and $T \subset \mathcal{T}$ the set of tasks in the plan. The initial plan contains temporal assertions tagged as *explained* meaning that the planner has to consider them as already true (see section "Building Plans and Strategies").

```
task TAKE_PICTURE(?obj, ?x, ?y)(t_start, t_end){
  ?obj in OBJECTS;
  hold(AT_ROBOT() : {{?x, ?y}}, (t_start, t_end));
  hold(PAN_TILT_UNIT_POSITION() : {{AT_MY_FEET}},
                                    (t_start, t_end));
  event(PICTURE(?obj, ?x, ?y) :
              ({{NONE}}, {{PICTURE_IDLE}}), t_start);
  hold(PICTURE(?obj, ?x, ?y) : {{PICTURE_IDLE}},
                                    (t_start, t_end));
  event(PICTURE(?obj, ?x, ?y) :
                ({{PICTURE_IDLE}}, {{DONE}}), t_end);
  uncontrollable (t_end - t_start) in [2, 4]; }
```

Figure 1: IxTeT specification of the task for taking one picture in the *Explore* domain.

Constraints on temporal variables are represented using a *Simple Temporal Network with Uncertainties* (STNU) (Vidal & Fargier 1999). Constraints on atemporal variables are represented using a general *Constraint Satisfaction Problem* (CSP) (Mackworth 1977). Furthermore, there is a mechanism (Trinquart & Ghallab 2001) to represent mixed constraints between atemporal and temporal variables. STNUs allow to specify bounds on differences between two temporal variables, including uncertain ones controlled by the environment and to check for the consistency of all the constraints. The uncontrollable duration of tasks are modeled as uncontrollable constraints in the STNU (Gallien & Ingrand 2006). For example, the task defined by IxTeT for a *Take Picture* action has an uncontrollable duration (see Figure 1).

TGA and Reachability Control Problem

Timed Game Automata (TGA) model has been introduced in (Maler, Pnueli, & Sifakis 1995) for the synthesis of timed controllers. It is now studied intensively in the computer aided verification domain and used to solve different kinds of problems (scheduling, testing, etc.). We first present quickly the simplest model of Timed Automata (Alur & Dill 1990) and then extend it to TGA.

Let $\mathcal{X}$ be a set of *clock variables*, which are real, positive variables with values evolving with the same rate as the time. The set $\mathcal{C}(\mathcal{X})$ of clock constraints over $\mathcal{X}$ contains formula ϕ generated by the following grammar: $\phi ::= x\#c \mid x - y\#c \mid \phi \wedge \phi$ where $c \in \mathbb{Z}$, $x, y \in \mathcal{X}$, and $\# \in \{<, \leq, \geq, >\}$. We denote by $\mathcal{B}(\mathcal{X})$ the subset of $\mathcal{C}(\mathcal{X})$ that uses only rectangular constraints of the form $x\#c$.

A *timed automaton* (TA) (Alur & Dill 1990) is a tuple $A = (\Sigma, \mathcal{X}, \mathcal{Q}, I, \Delta)$, where Σ is a finite set of actions, $\mathcal{X}$ is a finite set of clocks, and $\mathcal{Q}$ is a finite set of control states. The mapping $I : \mathcal{Q} \to \mathcal{B}(\mathcal{X})$ associates with each state $q \in \mathcal{Q}$ an *invariant* $I(q)$, also called staying condition. The set of transitions is $\Delta \subseteq \mathcal{Q} \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{X}} \times \mathcal{Q}$. A transition in Δ, also written $q \xrightarrow{g,\ell,r} q'$, specifies the source (q) and the target (q') states, the guard (g) which is a clock constraint to be satisfied when the transition is executed, the label of the transition (ℓ), and the subset of clocks to be assigned to 0 (r) as effect of the transition. This basic model of TA can be extended to allow state variables with finite values in guards, invariants, and assignments. A network of TA (nTA) is a finite set of TA evolving in parallel. This is the basic model of software tools dealing with the verification of TA.

A *configuration* of a TA (resp. nTA) is built from a state (resp. set of states) and a valuation of clock variables. From a given configuration, a TA can evolve (1) by executing a *discrete transition* in Δ which changes the state and the values of reset clocks, or (2) by executing a *timed transition*, i.e., by letting time pass in the current state and so incrementing all the clocks by the same value; the invariant of the state should be satisfied by the new values of the clocks. A *run* of a TA is a sequence of alternating timed and discrete transitions between configurations. The set of configurations for a TA is infinite, but Alur and Dill have shown that the number of classes of equivalent configurations w.r.t. the TA execution is finite. Such a class is called *region* and can be represented in $\mathcal{C}(\mathcal{X})$. *Symbolic configurations* are sets of regions. The notions of discrete transition, timed transition, and run can be extended to symbolic configurations.

A *timed game automaton* is a timed automaton $A = (\Sigma, \mathcal{X}, \mathcal{Q}, I, \Delta)$ where the set of actions Σ is split in two disjoint sets: Σ_c the set of controllable actions and Σ_u the set of uncontrollable actions. The notions of network of TA, run, and symbolic configuration are defined in a similar way for TGA. Figure 2 shows how the task of taking pictures in the *Explore* domain can be modeled using TGA. Transitions labeled by uncontrollable actions are represented using dashed arrows; state `q_PIC_NONE` is initial.

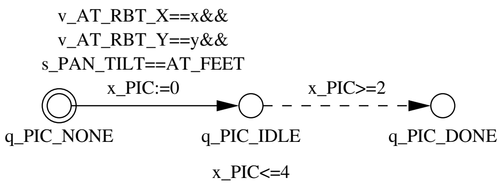

Figure 2: TGA model of the task for taking one picture in the *Explore* domain.

Given a TGA A and three symbolic configurations $\texttt{Init}$, $\texttt{Safe}$, and $\texttt{Goal}$, the *reachability control problem* or reachability game $\texttt{RG}(A, \texttt{Init}, \texttt{Safe}, \texttt{Goal})$ consists in finding a *strategy* f s.t. A starting from $\texttt{Init}$ and supervised by f stays in $\texttt{Safe}$ and enforces $\texttt{Goal}$. More precisely, a strategy is a partial mapping f from the set of runs of A starting from $\texttt{Init}$ to the set $\Sigma_c \cup \{\lambda\}$. For a finite run ρ, the strategy $f(\rho)$ may say (1) no way to win if $f(\rho)$ is undefined, (2) do nothing, just wait in the last configuration of ρ if $f(\rho) = \lambda$, or (3) execute the discrete, controllable transition labeled by ℓ in the last configuration of ρ if $f(\rho) = \ell$. A strategy f is *state-based* or *memory-less* whenever its result depends only on the last configuration of the run.

Building TGA from IxTeT Specifications

We briefly resume here the algorithm proposed in (Abdeddaïm, Asarin, & Sighireanu 2006) for building reachability control problems for a nTGA from IxTeT planning problems. In order to simplify the presentation, let us consider first a simple subset of IxTeT where: (1) state variables are not parameterized and theirs values are finite, (2) there are no constraints relating atemporal and temporal variables, (3) tasks are simple chronicles where only one state variable is modified by its events and these events are completely ordered. Then, translating IxTeT planning problems consists of the following steps: (a) for each state variable a generate a global, discrete variable s_a in the nTGA and a lock l_a protecting its modification; (b) for each task τ generate an automaton owning a clock $x_\tau \in \mathcal{X}$ as follows: (b.1) the initial transition acquires the lock l_{a_τ} of the state variable a_τ modified by the task and the final transition frees this lock; (b.2) for each value v taken by a_τ in τ generate a state $q_{\tau,v} \in \mathcal{Q}$; (b.3) for each event $event(a() : (?v_1, ?v_2), t)$ in τ generate a transition $q_{\tau,v_1} \xrightarrow{g,\ell,r} q_{\tau,v_2}$ in Δ where r assigns the global variable s_{a_τ} to the final value v_2 and resets the local clock x_τ; (b.4) from each persistence condition concerning a_τ generate guards, invariants, and the kind of label (controllable or uncontrollable) from the temporal constraints associated with the referenced time points; (b.5) from each persistence condition concerning state variables not modified by the task, generate additional guards and invariants for controllable transitions, and constraints in `Safe` for uncontrollable ones; (c) from the explained temporal assertions of the initial plan, generate the `Init` configuration; (d) from the persistence conditions of the set G of goals generate the `Goal` configuration.

The IxTeT model of the *Explore* domain satisfies, with two exceptions, the assumptions (1–3) above. The first exception concerns parameterized state variables of IxTeT. One solution is to flatten them into unparameterized state variables, so producing an exponential growth of the number of states. Fortunately, some modeling tools for TGA, e.g. UPPAAL-TIGA, allow specification of parameterized TGA: they do the flattening automatically. The second exception concerns the state variables denoting the position of the robot and the duration of the moving task of the robot. It is not possible to relate (real valued) delays with such variables. A solution to this problem is to pre-compute such relations, as detailed in the comparison section of this paper.

For the simple subset of IxTeT above, the size of the nTGA model obtained is linear in the size of the domain of (fully instantiated) state variables, the number of tasks, and in the number of temporal assertions. However, when parameterized state variables are used in the IxTeT specification, the complexity of the translation is exponential because of the flattening of state variables. For example, for the IxTeT specification of an *Explore* domain mission that takes four pictures in four different points and communicates two times with the satellite, we obtain a nTGA model with 31 states, 35 variables, 11 clocks, and 28 transitions.

Building Plans and Strategies

IxTeT Planner

The IxTeT planner (Ghallab & Laruelle 1994) is a plan space planner. The sketch of the algorithm is the following: (1) look for possible defaults in the current plan, if none are found, the plan is a solution plan (see Figure 3 for an example of such a plan); otherwise (2) chose one default and solve it (i.e. insert a chosen resolvant), if there is no resolvant, backtrack.

IxTeT produces plans that have an STNU dynamically controllable (Morris 2006) and a symbolic and numeric CSPs arc-consistent (Mackworth & Freuder 1985). The search algorithm will baktrack whenever this property is violated by the current plan.

A *default* is:

- A possible conflict between two temporal assertions: one “event" possibly at the same time than a “hold", two incompatible “holds" (different values, . . .)
- An unexplained temporal assertion: a temporal assertion cannot magically be true in a plan, but needs to be justified by another event already in the plan (i.e. explaining it).

A *resolvant* is:

- The insertion of some constraints in the plan.
- The insertion of “hold" and supporting constraints in order to make “causal links".

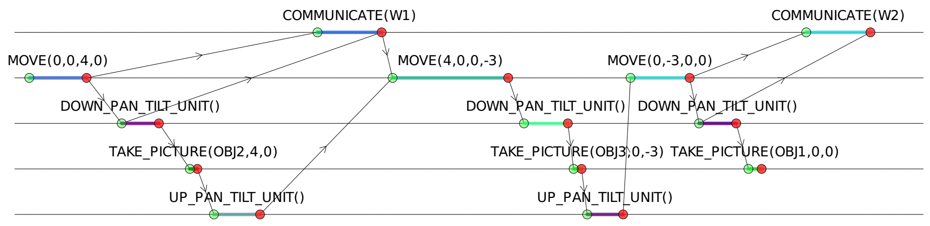

Figure 3: An example of solution plan found by IxTeT for the *Explore* domain.

- The insertion of a new task and a causal link.

IxTeT uses a depth first search (DFS) algorithm to find a solution plan. It employs a least commitment heuristic. The system computes all defaults and all their resolvants. Then for each resolvant a cost is computed. It is basically an estimate of the number of possible totally instantiated solution plans removed by the insertion of the resolvant.

The plans are both flexible thanks to the least commitment approach and temporally contingent thanks to the STNU. This is a very interesting property with respect to execution in the real world.

UPPAAL-TIGA

The algorithms used to solve reachability control problems on TGA are based on the exploration of the state space (i.e., all configurations) of the TGA. The classical algorithms for solving games explore backward (from the goal to the initial state) the state space. They suffer the usual "state explosion" problem of verification by model-checking. Recently, (Cassez *et al.* 2005) adapted the efficient forward/backward algorithm for untimed games of (Liu & Smolka 1998) to timed games. Moreover, they implemented it efficiently into a tool called UPPAAL-TIGA (Behrmann *et al.* 2007), which is an extension of the UPPAAL tool[1].

We briefly present here the principles of the algorithm published in (Cassez *et al.* 2005) and implemented, with additional optimisations, in UPPAAL-TIGA. For this, we need more notions on timed games. Let $\mathtt{RG}(A, \mathtt{Init}, \mathtt{Safe}, \mathtt{Goal})$ be a reachability game. A finite or infinite run ρ in A starting from Init is *winning* if it intersects at some point the Goal while staying in Safe configurations. A *maximal run* ρ of a TGA is either an infinite run or a finite run that satisfies either (i) last configuration of ρ is in Goal or (ii) the only possible next discrete actions from last configuration of ρ, if any, are uncontrollable actions. A strategy f is *winning* from a configuration s of A if all maximal runs obtained by executing A from s under the supervision of f are winning. A configuration s of A is winning if there exists a winning strategy f from s in A. We denote by $\mathcal{W}(RG)$ the set of winning configurations in A.

UPPAAL-TIGA takes as inputs (1) a network of TGA A where the initial configuration Init is fixed, and (2) a reachability control problem given by sets Safe and Goal. First, UPPAAL-TIGA explores forward A in order to find a winning run. During this exploration, UPPAAL-TIGA computes on the fly the product of TGA automata in A. At the present time, the forward exploration is done in a depth first search manner, without any heuristics to help the exploration. When a winning run is found, the forward exploration is suspended, and a backward, breadth first exploration is started in order to back-propagate the winning information to the previously explored configurations or their pending uncontrollable successors.

```
Strategy to win:
State: ( P1.q_PIC_DONE P2.q_PIC_NONE )
 When you are in true, take transition
  P2.q_PIC_NONE->P2.q_PIC_IDLE { 1, tau, x_PIC := 0 }
State: ( P1.q_PIC_NONE P2.q_PIC_NONE )
 When you are in true, take transition
  P1.q_PIC_NONE->P1.q_PIC_IDLE { 1, tau, x_PIC := 0 }
State: ( P1.q_PIC_DONE P2.q_PIC_IDLE )
 While you are in (P2.x_PIC<=4), wait.
State: ( P1.q_PIC_IDLE P2.q_PIC_IDLE )
 While you are in (P1.x_PIC<=4 && P2.x_PIC-P1.x_PIC<-2),
  wait.
State: ( P1.q_PIC_IDLE P2.q_PIC_NONE )
  When you are in (2<P1.x_PIC && P1.x_PIC<=4),
   take transition
   P2.q_PIC_NONE->P2.q_PIC_IDLE { 1, tau, x_PIC := 0 }
  While you are in (P1.x_PIC<=2), wait.
```

Figure 4: Example of a UPPAAL-TIGA strategy.

If the backward propagation returns to Init, the RG problem is solved and (Cassez *et al.* 2005) have shown that the explored configurations *are included in* (but not necessarily equal to) $\mathcal{W}(RG)$, the set of all winning configuration. If Init is not reached by the backward propagation, the forward exploration is resumed to find other winning runs.

When the problem is solved, a second exploration of the set of winning configurations builds a memory-less strategy f. This strategy is a *piece-wise function* giving a *deterministic* choice for actions to be done depending on the current configuration (values of clocks and variables). For example, Figure 4 presents the strategy built by UPPAAL-TIGA for the network of two (independent) TGA (P1 and P2) modeling the task of taking (different) pictures (see Figure 2) when the control problem asks to take them before 6 time units. (Local clocks and states are prefixed by the name of the task using a dotted notation and tau is the label for internal action.)

[1]UPPAAL is available at www.uppaal.com

Qualitative Comparison

Modeling Issues

The comparison between IxTeT and TGA models has been investigated on several examples and applications. From these studies, we have identified three significant differences that may be of interest for planning problems: (1) the capability of IxTeT to model mixed constraints, (2) the possibility to model periodic tasks with TGA, and (3) the fact that TGA can handle discrete choices of the environment. These three points are detailed here after.

Mixed constraints The first difference concerns the manipulation of state variables, as already mentioned in the translation subsection. State variables in IxTeT may belong to dense domains and can be constrained by timed variables. The kind of relations allowed depends on the underlying CSP system. For example, the task `MOVE` of Figure 5 relates the duration of moving (i.e., `t_end-t_start`) with the current and future position of the robot (i.e., `x`i, `y`i). In order to have tractable TGA models, state variables have to belong to finite domains and cannot be related with clocks. To relate clocks with state variables, such relations have to be approximated by (pre-computed) constants. For example, the task `MOVE` can be modeled by (1) fixing a set of positions to be observed (constant arrays `posx` and `posy`), (2) pre-computing the minimal and maximal duration of the navigation between each pair of positions (constant matrices `min_move` and `max_move` indexed by positions), and (3) using the pre-computed duration in clock constraints and invariants. Figure 6 provides the TGA model for the task `MOVE` for a fixed set of four positions. The discrete variable `crt` gives the index of the current position of the robot. When the robot is not moving (state `q_RB_STATUS_STL`, which is initial), it can choose to move in another position. We model it by a non-deterministic choice for values of variable `d`, the distance between the index of the current position and the index `nxt` of the next position. The navigation is done in state `q_RB_STATUS_STL` and its end is fixed by the environment (uncontrollable transition).

```
task MOVE (int x1, int y1, int x2, int y2)
  : (t_start, t_end) {
  // events and conditions saying that
  // the position @t_start is (x1,y1)
  // the position @t_end is (x2,y2)
  variable s in [0.13,0.25];
  variable dist in [0.5, +oo];
  // constraints approximating
  // dist by abs(x1-x2)+abs(y1-y2)
  variable duration;
  dist = s * duration;
  uncontrollable duration = t_end - t_start; }
```

Figure 5: Task `MOVE` represented using IxTeT constraints.

Periodic tasks The second difference concerns the specification of periodic tasks. IxTeT has to unfold periodic tasks on the full duration of the plan. For example, if a photography has to be taken every five minutes during one hour, IxTeT specification has to include explicitly 12 tasks. Instead,

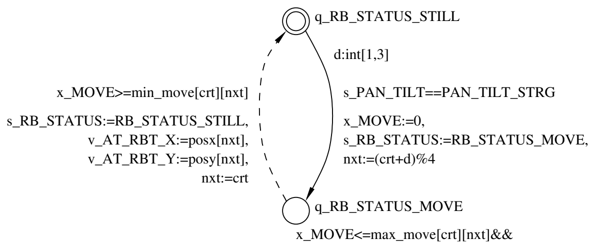
q_RB_STATUS_STILL
d:int[1,3]
x_MOVE>=min_move[crt][nxt]
s_PAN_TILT==PAN_TILT_STRG
s_RB_STATUS:=RB_STATUS_STILL,
v_AT_RBT_X:=posx[nxt],
v_AT_RBT_Y:=posy[nxt],
nxt:=crt
x_MOVE:=0,
s_RB_STATUS:=RB_STATUS_MOVE,
nxt:=(crt+d)%4
q_RB_STATUS_MOVE
x_MOVE<=max_move[crt][nxt]&&

Figure 6: Task `MOVE` represented using TGA.

TGA model allows natural specification (using loops) of periodic tasks.

Discrete choices of the environment The third difference concerns the specification of discrete choices of the environment. For example, consider a robot guide in a museum (the system) with a group of visitors (the environment). The robot has to present first picture P1, and this takes from 15 to 20 minutes. At the end of the presentation of P1, the visitors may choose, in maximum 3 minutes, between visiting picture P2 (which takes from 7 to 10 minutes) or picture P3 (which takes from 5 to 6 minutes).

The problem is to build a plan such that the full visit is done in 20 to 25 minutes. This type of problems cannot be represented by IxTeT: it is not possible to have one task with different possible outcomes.

In TGA (see Figure 7), two uncontrollable transitions (corresponding to each discrete choice of visitors) may outgo from the same state. Discrete choices of the environment are useful to model faults introduced by the environment: one choice is for the nominal behavior of the environment and the other(s) choice(s) is (are) for the faulty behavior. Then, it could be possible to obtain a fault tolerant plan/strategy, if it exists.

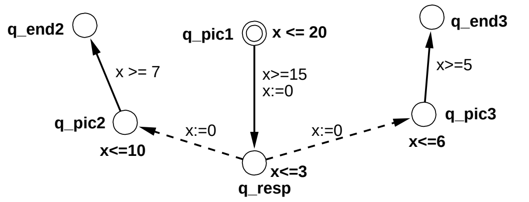

Figure 7: TGA model for the robot guide example.

Flexibility Issues

From the point of view of results produced (plans and strategies), there are two interesting differences between IxTeT and UPPAAL-TIGA.

Least commitment *versus* full instantiation The IxTeT algorithm for building plans does an unordered exploration of the defaults still in the plan and stops when no more defaults are present. The resolvants are as unrestrictive on the plan as possible, they will solve the default with a priori the least constraining solution. Moreover, a solution plan in IxTeT is not fully instantiated. It may contain state variables which are not instantiated, but still constrained in the CSPs. These two characteristics of IxTeT plans are known under the name of *least commitment*: the plan may contain choices that the controller has to determined at execution. Such property provides some flexibility to the controller. Furthermore, the IxTeT plans will not fail during execution if the uncontrollable durations remain between the given bounds, making them both flexible and contingent.

This least commitment property is not true for strategies generated by UPPAAL-TIGA, since the UPPAAL-TIGA strategies are deterministic. Indeed, during the forward exploration, the explored run fixes values for all state variables and an order between transitions. If the exploration is successful, this (fixed) order is chosen in the strategy.

For example, if the planning problem consists in planning two independent tasks taking photos, IxTeT planner builds a plan where there is no particular order between the beginning of these tasks. UPPAAL-TIGA fixes an order as shown by the strategy given on Figure 4: tasks `P1` is started first and then `P2` is started. In order to obtain some least commitment strategies, the algorithm of UPPAAL-TIGA shall be asked to compute all winning configurations $\mathcal{W}(RG)$ and not only the first subset found to include the initial configuration.

Conditional strategies Recall that strategies produced by UPPAAL-TIGA are deterministic, piece-wise functions saying which are the ways to stay in the set of winning configurations w.r.t. the current configuration. So several (deterministic) choices may be proposed by a strategy in a given configuration (see, e.g., Figure 4).

For example (Figure 8), let us consider the case of a paparazzi (controller) trying to take the picture of a star (environment) at the exit of its hotel before an Oscar ceremony. The star shall exit between 5 and 10 p.m. The photo is needed as soon as possible by the paparazzi's boss in order to be published either in the before dinner edition of its newspaper (i.e., before 8) or in the night edition of its entertainment channel (i.e., after 11).

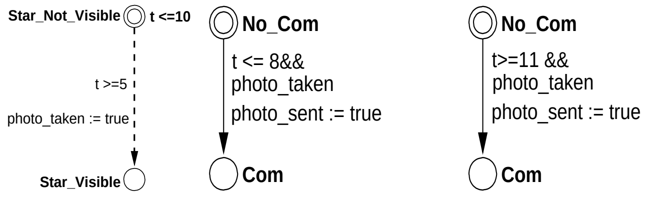

Figure 8: TGA model for the Paparazzi example.

For this example, UPPAAL-TIGA is able to generate a strategy (Figure 9) saying that if the star exits before 8 then take the photo and send it immediately, otherwise wait and send the photo after 11 as soon as possible.

A plan in IxTeT is a universal plan described by an STNU. Then it is not possible to use disjunction like the one needed to solve this example. If the problem is relaxed by removing the obligation to transmit as soon as possible, IxTeT finds a solution. However, the plans produced for the relaxed paparazzi example are not as efficient as strategies. For example, the plans will always transmit the picture for the night edition no matter the time of exit of the star, though the strategies will transmit the picture as soon as it is available.

This is a big advantage of strategies with respect to ex-

```
Strategy to win:
State: ( STAR.Star_Not_Visible COM1.No_Com COM2.No_Com )
  While you are in (t<=10), wait.
State: ( STAR.Star_Visible COM1.No_Com COM2.No_Com )
  When you are in (11<=t),
    take transition COM2.No_Com->COM2.Com
      { t >= 11 && photo_taken, tau, photo_sent := 1 }
  When you are in (5<=t && t<=8),
    take transition COM1.No_Com->COM1.Com
       { t <= 8 && photo_taken, tau, photo_sent := 1 }
  While you are in (8<t && t<11), wait.
```

Figure 9: Strategy to win for the Paparazzi example.

ecution facing temporal uncertainties. For example, in the *Explore* domain, the IxTeT plans may include unnecessary wait periods at execution time because the move has taken less time than the maximum expected duration. In that case, it may be possible to reschedule the plan in order to move back some tasks in the plan in order to take benefits from that opportunity, but that would take time to compute the new plan. Our experiments show that UPPAAL-TIGA strategies do not have this drawback.

Optimization Issues

Recall that the heuristic of IxTeT chooses to add a new task to the plan if and only if no other resolvants are possible. UPPAAL-TIGA does not have such heuristics nor causal links and it may generate strategies where tasks are done only to fill some waiting time. For example, in the *Explore* domain, while waiting the visibility window to communicate, the strategy produced by UPPAAL-TIGA contains movings of the cameras from the forward to the downward position and back, while it is clearly not required in the mission.

To avoid the planning of such useless actions by UPPAAL-TIGA, a solution is to introduce in the model additional constraints on the firing of transitions, depending on the domain. For example, in the *Explore* domain, the moving of the cameras shall be done only if the robot has to take a new picture or move again. Such constraints can be added directly on the model or by additional automata called observers, which monitor the behavior of the robot and prohibit some of its actions. These constraints can be derived automatically from the domain definition by looking at which tasks can establish new values for a state variable and which tasks expect a specific value for the same state variables.

There is a general solution to this issue for any kind of domain, but in a (possibly) less efficient manner. It consists in associating costs with some controllable discrete transitions and then asking for a strategy with a minimal total cost (i.e., minimal sum of all costs in runs). This kind of query is very expensive in computation time. A less expensive query is to ask for a total cost not exceeding some bound. The choice of the bound can be made by a binary search and trying to have reasonable computation times.

For example, to limit the number of useless moves of the cameras, a high cost can be associated to each controllable transition corresponding to such moves. To other controllable transitions, unitary costs can be added. Then, a shorter strategy can be obtained if we ask to solve the reachability control problem where the total goal cost is less than the cost needed for the moves of the cameras to take pictures plus some supplementary costs for other transitions.

Execution Issues

An IxTeT plan is not fully instantiated before execution. So, the execution is somewat equivalent to finding a solution of a CSP while maintaining it fully propagated, and this requires a significant amount of computation. The execution of the plan (Lemai & Ingrand 2004) consists in choosing a time point having all the necessary constraints on its predecessors satisfied (i.e. meaning that they are executed and the constraints between them are satisfied) and executing it (i.e. constraining it to occur at its execution time). Then, an incremental algorithm is used to efficiently propagate the STNU after the execution of each time point.

Executing UPPAAL-TIGA strategy consists in (1) collecting information about the current state of the controller, (2) finding the winning configuration containing this state, and (3) firing the transition proposed by the strategy for this state.

To sum up execution issues, a UPPAAL-TIGA strategy asks for less expensive execution procedures while an IxTeT plan allows to use some plan repair techniques (Lemai & Ingrand 2004) if something unexpected happens during the execution.

Quantitative Comparison

This section presents some experimental results obtained on the *Explore* domain by IxTeT and UPPAAL-TIGA. Note that UPPAAL-TIGA has been run on a TGA model obtained from the IxTeT specification using the principles of translation presented in this paper. However, one could obtain more compact models (and so more efficient results) by building a model from scratch. All the runs were made on a Pentium4 at 3.2 GHz and with 1GB memory. The evaluation problems include always two communications, at least one picture and the robot has to be back at its starting point at the end.

The first series of results (see Figure 10) compare the time for the two systems to find a solution for problems with solutions. We compare one IxTeT model to three different UPPAAL-TIGA models obtained as described in the section about optimization issues: the standard model, model with costs, and model with causal links. For IxTeT, the planning involves few bactracks. For UPPAAL-TIGA, the winning strategies obtained have a maximum duration close to the optimal one. In fact, UPPAAL-TIGA finds a plan faster if the maximum duration is close to the optimal one rather than a larger one. It comes from the fact that the number of possible explored states is smaller if the maximum duration is smaller. For the model with costs, UPPAAL-TIGA runs out of memory for 8 pictures because the number of states is multiplied by the number of possible values of the cost variable (45 in this experiment) and thus storing them uses too much memory. The modelisation of causal links is a way to cut some branches without solutions during the exploration of the model by UPPAAL-TIGA, thus better time performances are obtained.

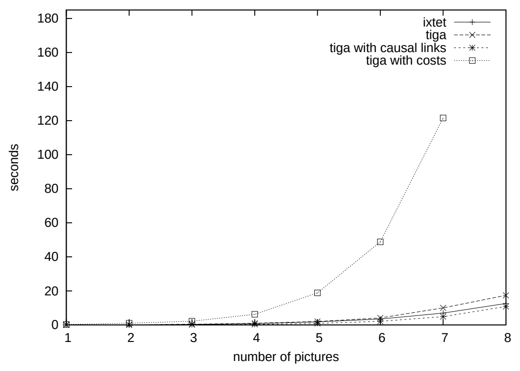

Figure 10: Duration in seconds to find a solution for problems with a a lot of solutions).

The next series of results (see Figure 11 and Figure 12) show how the systems behave when the problems have no solutions. Experiments are restricted to problems with 3 to 8 pictures. These results are important when considering the execution in the real world. Indeed, during the execution of a plan, the planner may be used to replan if some failures invalidate the current plan. It is important to have a planner answering as fast as possible to the question: “Is there a plan for all goals?”

IxTeT runs out of memory for the problem with 8 pictures because it does some node caching during the search in order to reduce the cost of backtracks and the cache becomes too big to fit in the memory when the full search space shall be explored.

These results show that UPPAAL-TIGA has a better behavior when the problem does not have solutions. Indeed, UPPAAL-TIGA is able to use the forward search algorithm to prune large part of the search space due to instantiation of all parameters and the maximum duration of the plan fixed by the goal. Instead, IxTeT needs to explore all the search space before concluding that no solution exists because it has no way to detect when a partially instantiated plan has no solution.

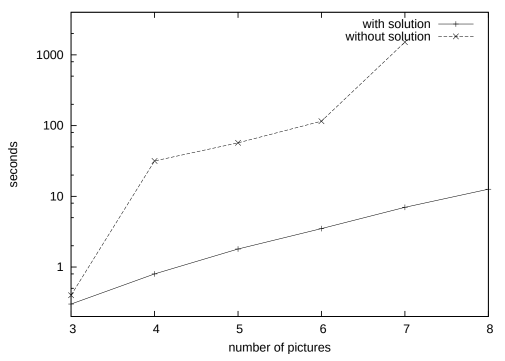

Figure 11: Duration in seconds for IxTeT to prove there is no solution (the scale is logarithmic).

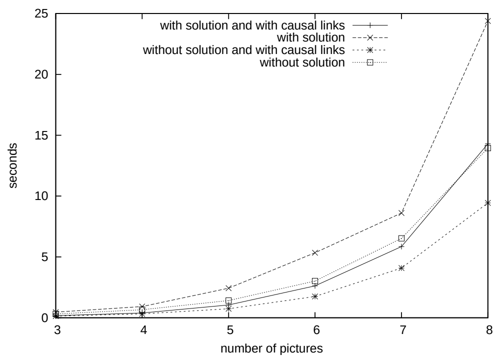

Figure 12: Duration in seconds for UPPAAL-TIGA to prove there is no solution.

Conclusion

Most embedded autonomous systems require some form of planning with temporal representation. Moreover, the plans/policies produced should be robust with respect to execution, i.e. leave some flexibility to the plan execution controller. Such flexibility can be obtained in various ways. One can allow events to occur in a temporal interval, or one can provide local repair mechanism, or the plan can propose multiple execution paths which all lead to the goal but with difference depending on the date of occurrence of some events. In our search to improve the planner on our mobile robots (on which we were using the CBTP IxTeT), we studied a completely different approach based on TGA and UPPAAL-TIGA which, among other things, can produce plans with a robust and more opportunistic execution. We show that a "classical" rover exploration problem can be modeled in UPPAAL-TIGA (we propose a general scheme to translate an IxTeT model to UPPAAL-TIGA) and produce plans with a temporally robust execution policy. We then compare the two approaches, qualitatively but also quantitatively, and show that each has its advantages and drawbacks. Thus, at this stage, it is not a clear cut which approach may be the best for a given problem and it is probably wiser to look at the features of the problem and the expected properties to decide which one to choose. However, having the possibility to use formal methods available in the timed automata community to prove some properties of a given plan model is also very encouraging with respect to the acceptability of automated planning in critical timed systems.

Acknowledgment

The authors would like to thank Alexandre David for his valuable support on UPPAAL-TIGA tool and its internals.

References

Abdeddaïm, Y.; Asarin, E.; and Maler, O. 2006. Scheduling with timed automata. *Theoretical Computer Sciences* 354(2).

Abdeddaïm, Y.; Asarin, E.; and Sighireanu, M. 2006. From IxTeT to timed games. Technical report, LIAFA.

Ai-Chang, M.; Bresina, J.; Charest, L.; Jónsson, A.; Hsu, J.; Kanefsky, B.; Maldague, P.; Morris, P.; Rajan, K.; and Yglesias, J. 2003. MAPGEN: Mixed initiative planning and scheduling for the Mars 03 MER mission. In *ISAIRAS*.

Alur, R., and Dill, D. 1990. Automata for modeling real-time systems. In *ICALP*.

Asarin, E., and Maler, O. 1999. As soon as possible: Time optimal control for timed automata. In *HSCC*.

Behrmann, G.; Cougnard, A.; David, A.; Fleury, E.; Larsen, K.; and Lime, D. 2007. Uppaal-Tiga: Time for playing games! In *CAV*.

Cassez, F.; David, A.; Fleury, E.; Larsen, K.; and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*.

Chien, S. 2006. Integrated AI in space: The autonomous sciencecraft on Earth observing one. In *AAAI*.

Cimatti, A.; Giunchiglia, F.; Giunchiglia, E.; and Traverso, P. 1997. Planning via model checking: A decision procedure for R. In *ECP*, 130–142.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8(4).

Gallien, M., and Ingrand, F. 2006. Controlability and makespan issues with robot action planning and execution. In *ICAPS workshop on Planning under Uncertainty and Execution Control for Autonomous Systems*.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *AIPS*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Jensen, R. M., and Veloso, M. M. 2000. Obdd-based universal planning for synchronized agents in non-deterministic domains. *JAIR* 13:189–226.

Khatib, L.; Muscettola, N.; and Havelund, K. 2001. Mapping temporal planning constraints into timed automata. In *TIME*.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI*.

Liu, X., and Smolka, S. 1998. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*.

Lowry, M. R.; Havelund, K.; and Penix, J. 1997. Verification and validation of ai systems that control deep-space spacecraft. In *ISMIS*, 35–47.

Mackworth, A., and Freuder, E. 1985. The complexity of some polynomial newtork consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25(1).

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In *STACS*.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *CP*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *ECP*.

Tripakis, S., and Altisen, K. 1999. On-the-fly controller synthesis for discrete and dense-time systems. In *FM*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI* 11(1).

Vidal, T. 2000. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *ICAPS*.