Generating Controllers for Flexible Plan Execution: a TGA approach

Andrea Orlandini[*] and **Alberto Finzi**[‡] and **Amedeo Cesta**[†] and **Simone Fratini**[†]

[*] ITIA-CNR, Via Bassini 15, I-20133 Milan, Italy
[‡] DSF "Federico II" University, Via Cinthia, I-80126 Naples, Italy
[†] ISTC-CNR, Via S.Martino della Battaglia 44, I-00185 Rome, Italy

Abstract

In temporal Planning and Scheduling (P&S) system, the synthesized plans may be temporally flexible and partially specified, therefore they need suitable executive systems for proper on-line execution. In general, instantiating and executing a temporally flexible plan is not an easy task due to constraint propagation and controllability issues. Previous works have tackled these problems by reasoning on the temporal constraints networks underlying the constraint-based plan representation often used by such systems. However, these issues can be addressed from a more abstract and general point of view deploying formal modeling and formal methods. In this work, we pursue such a second direction by presenting a formal method to synthesize a controller associated with a generated flexible temporal plan. Controller synthesis exploits Timed Game Automata (TGA) for formal modeling and UPPAAL-TIGA as a model checker. We present the method discussing both formal and empirical issues. The collected empirical results show the practical feasibility of the approach in a real-world robotic case study.

Introduction

In plan-based autonomous systems, robust plan execution in uncertain environments is a critical issue (see (Lemai and Ingrand 2004) and many others). Once a plan has been generated by a planner, its actual execution requires a suitable executive system capable of deciding on-the-fly how and when to execute the planned activities. This issue is particularly important in flexible temporal planning systems, where the generated plan is temporally flexible. Indeed, a flexible plan captures an envelope of possible temporal behaviors to be instantiated during the execution taking into account temporal/causal constraints and controllable/uncontrollable activities and events.

Previous works have tackled these issues in the Constraint-based Temporal Planning (CBTP) framework deploying specialized techniques based on temporal-constraint network. In particular, several authors (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005; Shah and Williams 2008) proposed the *dispatchable execution* approach where a compiler is to preprocess a flexible temporal plan into a dispatchable form which is then used by a plan dispatcher to online schedule the activities while guaranteeing constraints satisfaction.

In this paper, we propose to tackle the plan execution problem from a different point of view by introducing an alternative and novel technique based on formal modeling and control synthesis in Timed Game Automata (TGA). In particular, extending the approach proposed by (Cesta et al. 2010), here the generated flexible temporal plan and the dynamic domain are encoded into TGA models while a model checker is deployed to synthesize a real-time plan controller which guarantees plan execution along with other domain dependent properties. In contrast with dispatchable execution, here the synthesized controller works as an online generated dispatcher where all the decisions are already predefined, hence the on-line constraints check and propagation effort is not needed any more. This allows us to reduce the plan execution latency; on the other hand, beyond the planning effort, we have to consider the additional overhead of control synthesis which can be too expensive within a planning and execution cycle. In this paper, upon presenting the formal method, we explore also this issue discussing its practical applicability in a realistic robotic application scenario. The collected results show the feasibility of the approach and the relation between flexible plan generation and controller synthesis w.r.t. the complexity of the planning task and the generated plan.

In the literature, we can find analogous formal methods applied to plan synthesis and plan verification, but they do not address flexible temporal plan execution. Closely related to our work, in (Abdedaim et al. 2007) plan synthesis in TGA is proposed and contrasted w.r.t. flexible plan generation. In this case, the focus is on plan sysnthesis and TGA and CBTP are considered as alternative methods, instead, we connect them deploying control synthesis in TGA to generate a controller for a CBTP flexible plan. In (Goldman et al. 2002) timed automata are exploited for incremental verification within the plan generation process. In (Cesta et al. 2010) the authors propose UPPAAL-TIGA for flexible plan verification task, but they do not address plan control synthesis.

Plan-Based Robot Control

Our interest in plan-based software for autonomy is motivated by the GOAC project. We are developing a Goal Oriented Autonomous Controller (Ceballos et al. 2011) for the European Space Agency (ESA) integrating different software solutions. In particular: (a) a timeline-based deliberative layer which integrates a planner derived with the APSI

Software Platform (Cesta and Fratini 2008) and an executive that resembles the T-REX solution (Py, Rajan, and McGann 2010); (b) a functional layer which integrates G$^{en}{}_{o}$M and BIP as described in (Bensalem et al. 2010).

Independently on our current use, the work described in this paper is valid for any generic three layered control architecture (Gat 1997) that combines a physical layer, a functional layer, a deliberative layer, and a planning and scheduling system. The functional layer provides an abstraction of the physical system wrapping the controllers for the robotic devices (e.g., PTU, Camera, navigation, etc.). A generic deliberative layer is composed by a planning and scheduling module and an executive system. The planning and scheduling module is responsible for mission and task planning: given a set of mission goals, it generates temporal plans of actions to be delivered to the executive system. The executive system is responsible for plan monitoring, command dispatching and fault detection.

The Robotic Domain. In the paper, we use a running example taken from the GOAC project [1]. Let us consider a planetary rover equipped with a Pan-Tilt Unit (PTU), two stereo cameras (mounted on top of the PTU) and a communication facility. The rover is able to autonomously navigate the environment, move the PTU, take pictures and communicate images to a Remote Orbiter. A safe PTU position is assumed to be (*pan*, *tilt*) = $(0, 0)$. Finally, during the mission, the Orbiter may be not visible for some periods. Thus, the robotic platform can communicate only when the Orbiter is visible. The mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration. A possible mission action sequence is the following: navigate to one of the requested locations, move the PTU pointing at the requested direction, take a picture, then, communicate the image to the orbiter during the next available visibility window, put back the PTU in the safe position and, finally, move to the following requested location. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed. The rover must operate following some operative rules to maintain safe configurations and do not affect actions execution effectiveness. Namely, the following conditions must hold during the overall mission: **(C1)** While the robot is moving the PTU has to be in the safe position; **(C2)** The robotic platform can take a picture only if the robot is still in one of the requested location while the PTU is pointing at the related direction; **(C3)** Once a picture has been taken, the rover has to communicate the picture to the base station; **(C4)** While communicating, the rover has to be still; **(C5)** While communicating, the orbiter has to be visible. In real domains, it is not possible to determine in advance the actual execution duration for each task. Thus, termination commands of each robot task are to be considered as uncontrollable to the executive system.

[1]Thanks to Felix Ingrand and Lavindra De Silva from LAAS-CNRS for the time spent to explain us the details of their robotic platform.

Timeline-based planning and execution

Timeline-based planning is an approach to temporal planning that has been applied in the solution of several real world problems – e.g., (Muscettola 1994). The approach pursues a general idea that planning and scheduling for controlling complex physical systems consist in the synthesis of desired temporal behaviors (or *timelines*).

State variables and timelines. According to this paradigm a domain is modeled as a set of features with associated set of temporal functions on a finite set of values. The time varying features are called *multi-valued state variables* as in (Muscettola 1994). As in classical control theory, the evolution of the features are described by some causal laws and limited by domain constraints. These are specified in a *domain specification*. The task of a planner is to find a sequence of decisions that bring the timelines into a final desired set always satisfying the domain specification and special conditions called *goals*. We assume that the temporal features – represented by the state-variables – have a finite set of possible values assumed over temporal intervals. The temporal evolutions are sequences of operational states. Causal and temporal constraints specify which value transitions are allowed, the duration of each valued interval (i.e., how long a given operational status can be maintained) and synchronization constraints between different state variables.

More formally, a state variable is defined by a tuple $\langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$ where: (a) $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *values*; (b) $\mathcal{T} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$ is the *value transition* function; (c) $\mathcal{D} : \mathcal{V} \rightarrow \mathbb{N} \times \mathbb{N}$ is the *value duration* function, i.e. a function that specifies the allowed duration of values in $\mathcal{V}$ (as an interval $[lb, ub]$). (b) and (c) specify the operational constraints on the values in (a). Given a state variable, its associated timeline is represented as a sequence of values in the temporal interval $\mathcal{H} = [0, H)$. Each value satisfies previous (a-b-c) specifications and is defined on a set of not overlapping time intervals contained in $\mathcal{H}$.

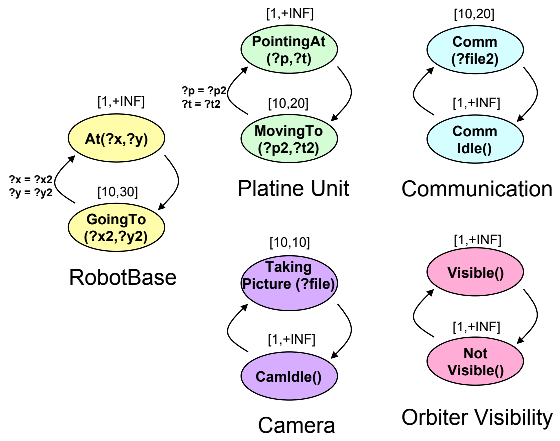

Figure 1: *Value transitions for state variables describing the robotic platform activities (Temporal durations in seconds)*

Timeline specification for the robotic domain. To obtain a timeline-based specification of our robotic domain, we consider two types of state variables: *Planned State Variables* to represent timelines whose values are decided by the planning agent, and *External State Variables* to represent timelines whose values over time can only be observed. Planned state variables are those representing time varying features like the temporal occurrence of navigation, PTU, camera and communication operations. We use four of such state variables, namely the *RobotBase*, *PTU*, *Camera* and *Communication*. In Figure 1, we detail the values that can be assumed by these state variables, their durations and the legal value transitions in accordance with the mission requirements and the robot physics. Additionally, one external state variable represents contingent events, in particular the communication opportunities. The *Orbiter Visibility* state variable maintains the visibility of the orbiter. The allowed values for this state variable is *Visible* or *Not-Visible* and are set as an external input.

The robotic platform can be in a certain position (*At(x,y)*) or moving to a certain destination (*GoingTo(x,y)*). The PTU can assume a *PointingAt(pan,tilt)* value if pointing a certain direction, while, when moving, it assumes a *MovingTo(pan,tilt)*. The camera can take a picture of a given object in a position $\langle x, y \rangle$ with the PTU in $\langle pan, tilt \rangle$ and store it as a file in the on-board memory (*TakingPicture(file-id,x,y,pan,tilt)*) or be idle (*CamIdle()*). Similarly, the communication facility can be operative and dumping a given file (*Communicating(file-id)*) or be idle (*ComIdle()*).

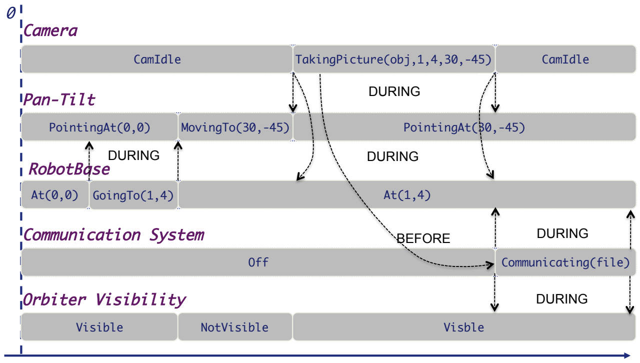

Figure 2: *An example of timeline-based plan with synchronizations.*

Representing domain causality. Domain operational constraints are described by means of *synchronizations*. A synchronization models the existing temporal and causal constraints among the values taken by different timelines (i.e., patterns of legal occurrences of the operational states across the timelines). Figure 2 exemplifies the use of synchronizations in our case study domain. The following synchronizations are represented in the figure: the *PointingAt(0,0)* value must occur during a *GoingTo(x,y)* value (**C1**); the *At(x,y)* and *PointingAt(pan,tilt)* values must occur during a *TakingPicture(pic,x,y,pan,tilt)* value (**C2**); the *Communicating(pic)* must occur after a *TakingPicture(pic,x,y,pan,tilt)* (**C3**); the *At(x,y)* value must occur during a *Communicating(file)* (**C4**); the *Visible* value must occur during a *Communicating(file)* (**C5**). (C1) and (C4) represent safety conditions: when moving or communicating the rover must be in a safe configuration (PTU unit in (0,0) when moving or not moving when communicating). (C2) and (C5) represent temporal synchronizations among different activities (to take a picture the rover must be in the proper place and configuration in the right time (C2) and dumps must occur when the orbiter is visible (C5)). (C3) describes a pure cause-effect relationships between two activities: pictures must be dumped once stored. In addition to those synchronization constraints, the timelines must respect transition constraints among values and durations for each value specified in the domain (see again Figure 1).

Timeline-based planning. *Planning goals* are expressed as desired timeline values in temporal intervals; the task of the planner is to build a set of timelines that describe valid sequences of values that achieve the desiderata. Hence, a *plan* is a set of *timelines*, that is, a sequence of state variable values, a set of ordered transition points between the values, and a set of distance constraints between transition points. When the transition points are bounded by the planning process (lower and upper bounds are given for them) instead of being exactly specified, we refer to the timeline as *time flexible* and a *flexible plan* is the plan resulting from a set of flexible timelines. A flexible plan defines a set of admissible temporal behaviors. Considering a partial horizon H' (with $H' < H$), the same flexible plan defines a set of *partial* temporal behaviors $\mathcal{PB}$. A *flexible plan* $\mathcal{P} = \{TL_1, ..., TL_n\}$ is defined over a given horizon $[0, H]$. The process of *solution extraction* from a plan is the process of computing (if exists) a *valid* and completely specified set of timelines from a given set of time-flexible timelines. A solution is *valid* with respect to a domain theory if every temporal occurrence of a reference value implies that the related target values hold on target timelines presenting temporal intervals that satisfy the expected relations.

Plan execution. During plan execution, the plan, or segment of it, is under responsibility of the executive system that forces value transitions over the timelines dispatching commands to the functional layers while continuously accepting observations and, thus, monitoring the plan execution. Additionally, not all the value transitions are under responsibility of the executive, but event exists that are under control of *nature*. As a consequence, an executive cannot completely predict the behavior of the controlled physical system because the duration of certain processes or the timing of exogenous events is outside of its control. In such cases, the values for the controllable state variables should be chosen so that they do not constrain uncontrollable events. This is the *controllability problem* defined, e.g., in (Vidal and Fargier 1999) where *contingent* and *executable* processes are distinguished. The contingent processes are not controllable, hence with uncertain durations, instead the executable processes are started and ended by the executive system. Controllability issues underlying a plan representation have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) representation in (Vidal and Fargier 1999) where basic formal notions are given for *dynamic* controllability (see also (Morris and Muscettola 2005)). In (Cesta et al. 2010) these notions have been extended to the timeline-based framework.

Since the executive system has to safely execute a flexible plan while taking into account the controllability problem, a plan controller is needed to execute a flexible temporal plan. More formally, a *plan controller* $\mathcal{C}$ is a partial function from the set of partial behaviors $\mathcal{PB}$ and possible horizons to the set of controllable values for state variables plus a special action λ representing the *wait* action as for TGA (see the following section), $\mathcal{C} : \mathcal{PB} \times \mathbb{N} \rightarrow \mathcal{V}_1 \cup ... \cup \mathcal{V}_n \cup \{\lambda\}$.

Timed Game Automata and Controllers

Timed game automata (TGA) have been introduced in (Maler, Pnueli, and Sifakis 1995) to model control problems on timed systems (Cassez et al. 2005), in this section we briefly recall some definitions that we shall use in the rest of the paper.

Definition 1 *A* **Timed Game Automaton** *is a tuple* $\mathcal{A} = (Q, q_0, \text{Act}, X, \text{Inv}, E)$ *where: Q is a finite set of* locations*;* $q_0 \in Q$ *is the* initial location*;* Act *is a finite set of* actions *split in two disjoint sets,* Act_c *the set of* controllable *actions and* Act_u *the set of* uncontrollable *actions; X is a finite set of a nonnegative, real-valued variables called clocks;* $\text{Inv} : Q \rightarrow B(X)$ *is a function associating to each location* $q \in Q$ *a constraint* $\text{Inv}(q)$ *(the* invariant *of q);* $E \subseteq Q \times B(X) \times \text{Act} \times 2^X \times Q$ *is a finite set of* transitions*. Where* $B(X)$ *is the set of constraints in the form* $x \sim c$*, where* $c \in \mathbb{Z}$, $x, y \in X$*, and* $\sim \in \{<, \leq, \geq, >\}$*. We write* $q \overset{g,a,Y}{\rightarrow} q' \in E$ *for (q, g, a, Y, q')* $\in E$*.*

A *state* of a TGA is a pair $(q, v) \in Q \times \mathbb{R}^X_{\geq 0}$ that consists of a discrete part and a valuation of the clocks (i.e., a value assignment for each clock in X). An *admissible* state for a $\mathcal{A}$ is a state (q, v) s.t. $v \models \text{Inv}(q)$. From a state (q, v) a TGA can either let time progress or do a discrete transition and reach a new state. A *time transition* for $\mathcal{A}$ is 4-tuple $(q, v) \overset{\delta}{\rightarrow} (q, v')$ where $(q, v) \in S$, $(q, v') \in S$, $\delta \in R_{\geq 0}$, $v' = v + \delta$, $v \models \text{Inv}(q)$ and $v' \models \text{Inv}(q)$. That is, in a time transition a TGA does not change location, but only its clock values. Note that all clock variables are incremented by the same amount δ in valuation v'. This is why variables in X are named *clocks*. Accordingly, δ models the *elapsed time* during the time transition. A *discrete transition* for $\mathcal{A}$ is 5-tuple $(q, v) \overset{a}{\rightarrow} (q', v')$ where $(q, v) \in S$, $(q', v') \in S$, $a \in \text{Act}$ and there exists a transition $q \overset{g,a,Y}{\rightarrow} q' \in E$ s.t. $v \models g$, $v' = v[Y]$ and $v' \models \text{Inv}(q')$. In other words, there is a discrete transition (labeled with a) from state (q, v) to state (q', v') if the clock values (valuation v) satisfy the *transition guard* g and the clock values after resetting the clocks in Y (valuation v') satisfy the invariant of location q'. Note that an admissible transition always leads to an admissible state and that only clocks in Y (reset clocks) change their value (namely, to 0). A *run* of a TGA $\mathcal{A}$ is a finite or infinite sequence of alternating time and discrete transitions of $\mathcal{A}$. We denote with $\text{Runs}(\mathcal{A}, (q, v))$ the set of runs of $\mathcal{A}$ starting from state (q, v) and write $\text{Runs}(\mathcal{A})$ for $\text{Runs}(\mathcal{A}, (q, \vec{0}))$. If ρ is a finite run, we denote with $\text{last}(\rho)$ the last state of run ρ and with $\text{Duration}(\rho)$ the sum of the elapsed times of all time transitions in ρ. A *network* of TGA (nTGA) is a finite set of TGA evolving in parallel with a CCS style semantics for parallelism. Namely, at any time, only one TGA in the network can change location, unless a synchronization on labels takes place. In the latter case, the two automata synchronizing on the same label move together. Note that time does not elapse during synchronizations.

Given a TGA $\mathcal{A}$ and three symbolic configurations *Init*, *Safe*, and *Goal*, the *reachability control problem* or reachability game $RG(\mathcal{A}, \textit{Init}, \textit{Safe}, \textit{Goal})$ consists in finding a *strategy* f such that $\mathcal{A}$ starting from *Init* and supervised by f generates a winning run that stays in *Safe* and enforces *Goal*. A strategy is a partial mapping f from the set of runs of $\mathcal{A}$ starting from *Init* to the set $\text{Act}_c \cup \{\lambda\}$ (λ is a special symbol that denotes ”do nothing and just wait”). 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 ρ if $f(\rho) = \lambda$, or (3) execute the discrete, controllable transition labeled by l in the last configuration of ρ if $f(\rho) = l$. The restricted behavior of a TGA $\mathcal{A}$ controlled with some strategy f is defined by the notion of *outcome*. The outcome $Outcome(q, f)$ is defined as the subset of $Runs(\amalg, \mathcal{A})$ that can be generated from q executing the uncontrollable actions in Act_u or the controllable actions provided by the strategy f. A *maximal run* ρ is either an infinite run or a finite run that satisfies either i) $last(\rho) \models Goal$ or ii) if $\rho \overset{a}{\rightarrow}$ then $a \in Act_u$ (i.e. the only possible next discrete actions from $last(\rho)$, if any, are uncontrollable actions). A strategy f is a *winning strategy* from q if all maximal runs in $Outcome(q, f)$ are in $WinRuns(q, \mathcal{A})$. A state q in a TGA $\mathcal{A}$ is *winning* if there exists a winning strategy f from q in $\mathcal{A}$. We denote by $W(\mathcal{A})$ the set of winning states of $\mathcal{A}$.

The encoding as nTGA model

As said before, TGAs allow to model real-time systems and controllability problems representing uncontrollable activities as *adversary moves* within a game between the controller and the environment. Following the same approach presented in (Cesta et al. 2010), flexible timeline-based plan verification can be performed by solving a Reachability Game using UPPAAL-TIGA.

To this end, we compile flexible timeline-based plans, state variables, and domain theory descriptions into set of TGA (nTGA). This is obtained with the following steps.

First, a flexible timeline-based plan $\mathcal{P}$ is mapped into a nTGA *Plan*. Each timeline is encoded as a sequence of locations (one for each timed interval), while transition guards and location invariants are defined according to (respectively) lower and upper bounds of flexible timed intervals.

```
process RobotBase() {

  state GoingTo {clockRobotBase <= 30}, At;

  init At;

trans
  GoingTo -u-> At {guard clockRobotBase >= 10;
      sync pulse_RobotBase_At?; assign clockRobotBase := 0,
      RobotBaseGoingTo := false, RobotBaseAt := true; },

  At -> GoingTo {guard clockRobotBase >= 1;
     sync pulse_RobotBase_GoingTo?; assign clockRobotBase := 0,
     RobotBaseGoingTo := true, RobotBaseAt := false; };
}
```

Figure 3: *TGA specification for the “RobotBase” state variable*

Then, the related set of state variables *SV* is mapped into a nTGA *StateVar*. Basically, we define a one-to-one mapping from state variables descriptions to TGA (e.g. in Fig. 3 we can fined the UPPAAL-TIGA specification for the *RobotBase* state variable). In this encoding, value transitions are partitioned into controllable and uncontrollable.

Finally, we introduce an *Observer* automaton to check for value constraints violations and synchronizations violations. In particular, we have two locations: an Error location, to state constraint/synchronization violations, and a Nominal (OK) location, to state that the plan behavior is correct. The *Observer* is defined as fully uncontrollable. An excerpt of the TGA monitor specification is provided in Fig. 4.

```
process monitor() {  state OK,ERR;
  init OK;

  trans
    OK -u-> ERR {guard (stepRobotBase == 0) and not (RobotBaseAt);},
    OK -u-> ERR {guard (stepRobotBase == 1) and not (RobotBaseGoingTo);},
    OK -u-> ERR {guard (stepRobotBase == 2) and not (RobotBaseAt);},
    ...

    // -- DT --
    OK -u-> ERR {guard (clockRobotBase > 0) and (clockPlatine > 0)
          and  (RobotBaseGoingTo) and not (PlatinePointingAt);  },
    OK -u-> ERR {guard (clockCamera > 0) and (clockRobotBase > 0)
          and  (CameraTakingPicture) and not (RobotBaseAt);  },
    ...

    ERR -u-> ERR { };
}
```

Figure 4: *The TGA specification of the "Observer"*

The nTGA $\mathcal{PL}$ composed of the set of automata *StateVar* $\cup$ *Plan* $\cup \{\mathcal{A}_{Obs}\}$ models a flexible plan along with state variables and domain theory descriptions. The following theorem has been demonstrated (Cesta et al. 2010):

Theorem 1 *The nTGA $\mathcal{PL}$ describes all and only the behaviors implemented by the flexible plan $\mathcal{P}$.*

Considering a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of the initial locations of each automaton in $\mathcal{PL}$, *Safe* is the Observer's OK location, and *Goal* is the set of goal locations, one for each automaton in *Plan*, the following theorem has been demonstrated:

Theorem 2 *Given $RG(\mathcal{PL}$,* Init, Safe, Goal*) defined considering* Init, Safe *and* Goal *as above, winning the game implies plan validity for $\mathcal{P}$. (Cesta et al. 2010).*

Thus, to perform plan verification, the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) defined above can be solved by means of UPPAAL-TIGA. If there is no winning strategy, UPPAAL-TIGA provides a counter strategy for the opponent (i.e., the environment) to make the controller lose. Then, to solve the reachability game, we ask UPPAAL-TIGA to check the CTL formula $\Phi = A$ *[Safe U Goal]* in $\mathcal{PL}$. In fact, this formula states that along all the possible evolutions, $\mathcal{PL}$ remains in *Safe* states until *Goal* states are reached. Thus, if the solver verifies the above property, then the flexible temporal plan is valid. Whenever the flexible plan is not verified, UPPAAL-TIGA produces an execution strategy showing one temporal evolution that leads to a fault.

Synthesizing Controllers

In this section, we provide some formal definitions for plan controller synthesis as a direct extension of the theorems presented above.

Theorem 1 defines a one-to-one mapping between flexible temporal behaviors over $[0, H]$ defined by $\mathcal{P}$ and the automata behaviors defined by $\mathcal{PL}$. This property can be directly extended to partial plans. Indeed, since Theorem 1 holds from any horizon H, for each partial temporal behavior $pb \in \mathcal{PB}$ defined over $H' < H$, there exists a unique run ρ_{pb} of $\mathcal{PL}$ such that ρ_{pb} represents the temporal behavior pb over the same horizon H'. That is, ρ_{pb} of $\mathcal{PL}$ represents the same valued intervals sequence in $\mathcal{P}$ limited to H' and *Duration*(ρ_{pb}) is exactly the horizon H'.

Analogously, by extending Theorem 1 to partial plans, the winning strategy obtained as a side effect of the verification process represents a flexible plan controller that achieves the planning goals keeping the dynamic controllability during the overall plan execution. More formally, a plan controller $\mathcal{C}_f$ derived from a winning strategy f can be defined as follows.

Definition 2 *Given the reachability game $RG(\mathcal{PL}$,* Init, Safe, Goal*) defined as above, the winning strategy f generated by UPPAAL-TIGA defines a plan controller $\mathcal{C}_f$ as follows: for each partial behavior $pb \in \mathcal{PB}$ over H', $\mathcal{C}_f(pb, H') = f(\rho_{pb})$ where each action $a \in Act_c \cup \{\lambda\}$ represents the associated values in $\mathcal{V}_1 \cup ... \cup \mathcal{V}_n \cup \{\lambda\}$, otherwise $\mathcal{C}_f(pb, H')$ is undefined.*

As a consequence, the following theorem holds.

Theorem 3 *The controller $\mathcal{C}_f$ defined in Definition 2 (i) correctly executes the plan $\mathcal{P}$ reaching the given planning goals and (ii) maintains the dynamic controllability property during the plan execution.*

Moreover, it is also possible to define optimized controllers for flexible plans. Indeed, given a fixed temporal interval $[u, g]$ along with a reachability game, UPPAAL-TIGA is able to generate a winning strategy f^* within that interval which minimize the plan execution duration (Cassez et al. 2005). Since a flexible plan is associated with a planning horizon $[0, H]$, an optimized controller can be generated with $[u, g]$ = $[0, H]$. This allows to conclude the following.

Theorem 4 *Given the reachability game $RG(\mathcal{PL}$,* Init, Safe, Goal*) defined as above within the temporal interval [u,g] = [0,H], the winning strategy provided by UPPAAL-TIGA f^* is time optimal and, for Theorem 3, also the derived controller $\mathcal{C}_{f^*}$ is time optimal.*

Empirical Results

In this section, we discuss the practical feasibility of the approach using real problem instances for the robotic case study presented above. In particular, we introduce different planning/execution scenarios obtained by varying: the number of pictures to be taken and the plan horizon; the allowed temporal flexibility; the number of communication opportunities. These settings are defined as follows:

– Number of Pictures and Horizon. We consider problem instances with an increasing number of requested pictures (from 1 up to 5). At the same time, we consider flexible plans with a horizon length ranging from 150 to 550 seconds.

– Flexibility. For each uncontrollable activity (i.e., *GoingTo*, *MovingTo*, *TakingPicture*, and *Communicating*), we set a minimal duration, but we allow temporal flexibility on the activity termination, namely, the end of each activity has a tolerance ranging from 0 to 20 seconds. This temporal interval represents the degree of temporal flexibility/uncertainty that we introduce in the system.

– Number of visibility windows. We set from 1 to 4 visibility windows that can be exploited to communicate pictures. Notice that an increasing number of communication opportunities raises the complexity of the planning problem with a combinatorial effect.

Among all the generated problem instances, the hardest are the ones with higher number of required pictures, higher temporal flexibility, and higher number of visibility windows. As a planner, we use the OMPS (Open Multi-Component Planner and Scheduler (Fratini, Pecora, and Cesta 2008)), a CBTP Domain Independent Planner. In these scenarios, we analyzed the performance of our method considering: model generation, controller synthesis, and plan execution. We run our experiments on a MacBook Pro endowed with an Intel Core i5 (2.5GHz) processor and 4GB RAM. In the following we illustrate the collected empirical results (the reported timings are in seconds).

Model Generation. As a preliminary step of our evaluation, we consider the cost of generating the UPPAAL-TIGA model associated with the planning task. The dimension of the generated model is given in terms of number of generated states and bytes dimension of the files. The results are illustrated in Table 1. For all these configurations, we observe that the generation process is very fast, taking less than 200ms, while the dimension of the generated model gradually grows with respect to the dimension of the flexible plan in terms of both plan length and number of visibility windows. Temporal flexibility does not affect the dimension of the generated models. Thus, we can conclude that model generation is not a critical step for our method.

Table 1: *Size of generated models (bytes and number of states) with respect to the plan length. Flexibility does not affect the size.*

		1 wind	2 wind	3 wind	4 wind
pic 1	bytes	8108	8108	8671	8960
H 150	nr. of states	29	29	33	35
pic 2	bytes	10094	10370	10674	10936
H 250	nr. of states	39	39	43	45
pic 3	bytes	13051	13326	13603	13892
H 350	nr. of states	53	53	57	59
pic 4	bytes	14102	14378	14655	14943
H 450	nr. of states	59	63	65	69
pic 5	bytes	18151	16402	16678	16967
H 550	nr. of states	69	70	73	75

Controller Synthesis. We analyze the cost of controller synthesis with respect to the cost of Planning and the cost of Plan verification (i.e. dynamic controllability check). Furthermore, we assess the dimension of the generated controller in terms of number of rules defined by the UPPAAL-TIGA winning strategy and number of kilobytes to store the strategy. Similar data are collected for optimized controller generation.

The planning costs are collected in Table 2. Here, the planner performance decreases with increasing communication windows and temporal flexibility. In particular, simple instances (i.e., 1 or 2 communication windows, see Tables 2(a) and 2(b)) are solved in few seconds, while the hardest ones (i.e., 4 communication windows and more than three required pictures, see Tables 2(c) and 2(d)) require and additional planning effort. Actually, the planner is not able to solve some of the instances due to memory limit (N/A in Table 2(d)).

Table 2: *Plan generation cost varying the number of required pictures, the number of visibility windows and the temporal flexibility.*

(a)

1 Comm. Window			
pic	0s flex	10s flex	20s flex
1	2,105	2,319	2,242
2	2,447	2,593	2,295
3	3,046	2,981	2,936
4	2,958	2,909	2,935
5	6,023	4,108	4,063

(b)

2 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	2,247	2,225	2,178
2	2,816	2,356	2,404
3	4,461	3,152	4,337
4	3,276	5,872	4,451
5	4,058	4,040	4,118

(c)

3 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	2,275	2,263	2,214
2	3,208	3,829	3,069
3	6,025	5,953	5,940
4	17,247	20,307	26,944
5	72,698	123,386	252,386

(d)

4 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	2,242	2,194	2,261
2	2,306	2,285	2,261
3	6,237	14,285	27,677
4	193,204	237,139	N/A
5	54,288	N/A	N/A

The results collected for dynamic controllability check (see Table 3) and strategies generation (see Table 4) show a quite different behavior. Interestingly, for hard problem instances flexible plan verification and strategy generation are very fast (Tables 3(d) and 4(d)). On the other hand, with simpler instances (Tables 3(a) and 4(a)) we do not observe the expected improvement in performance. This is mainly due to the fact that simple planning problems are associated with few constraints to be considered, hence our planner can generate highly flexible temporal plans. However, this flexibility provides a wide search space to the verification tool reducing its performance. In contrast, harder planning problems lead the planner to produce flexible plans that are strongly constrained, i.e., with a lower degree of flexibility. This simplifies the UPPAAL-TIGA task which can check and generate strategies more quickly (see again Tables 3(d) and 4(d)). Also in Table 6, the same behavior is shown in which more flexible plans are associated with more complex controller (i.e., more rules and, then, more kbytes), while more constrained plans require simpler controllers. Indeed, this is an expected behavior of the verification tool. In fact, the more non-determinism, the harder it is for UPPAAL-TIGA to generate strategies.

Thus, an additional effort during the planning phase usually reduces the cost of plan verification and control synthesis and, vice-versa, simpler planning problems are usually associated with more complex controller synthesis tasks.

In particular, for the robotic case study, if we contrast the controller generation time w.r.t. the planning horizon length (assuming a comparable time available for planning), we can see that taking apart the hardest instances (5 pictures, and 3 or 4 pictures with less than 3 visibility windows), all the other cases are treatable. On the other hand, when we consider the control synthesis cost overhead with respect to the plan generation cost, we observe that, with few pictures (e.g., 1 or 2 required picture), for all the visibility windows

Table 3: *Dynamic Controllability verification cost.*

(a)

1 Comm. Window			
pic	0s flex	10s flex	20s flex
1	0,062	0,074	0,088
2	2,294	3,964	4,611
3	18,131	36,980	53,664
4	61,115	106,637	120,458
5	167,279	307,876	372,972

(b)

2 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,056	0,074	0,085
2	0,115	0,200	0,208
3	4,184	6,815	8,363
4	19,053	36,063	38,971
5	70,011	126,452	153,978

(c)

3 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,018	0,019	0,018
2	0,030	0,030	0,029
3	0,382	0,674	0,870
4	2,576	5,025	6,789
5	20,760	36,971	36,201

(d)

4 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,019	0,019	0,018
2	0,031	0,030	0,029
3	0,396	0,640	0,760
4	0,190	0,265	N/A
5	19,184	N/A	N/A

Table 4: *Strategies generation cost for both optimized and non optimized cases.*

(a)

1 Comm. Window			
pic	0s flex	10s flex	20s flex
1	0,089	0,153	0,266
2	7,067	12,636	15,142
3	57,172	121,62	185,806
4	202,023	361,326	410,104
5	834,934	892,232	923,345
Optimal			
1	0,289	0,356	0,485
2	13,900	25,161	28,953
3	138,257	229,638	263,062
4	396,050	607,506	965,803
5	1521,22	1634,23	2526,90

(b)

2 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,090	0,153	0,269
2	0,257	0,612	1,026
3	12,859	20,949	26,671
4	60,071	113,865	134,046
5	224,311	400,734	500,811
Optimal			
1	0,290	0,470	0,795
2	0,767	1,213	1,755
3	27,489	51,933	61,005
4	151,161	180,928	314,222
5	741,078	861,078	880,952

(c)

3 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,021	0,022	0,021
2	0,043	0,043	0,041
3	1,079	1,945	2,702
4	7,655	15,900	22,866
5	58,318	121,208	117,292
Optimal			
1	0,022	0,022	0,023
2	0,064	0,084	0,077
3	1,906	3,150	4,840
4	18,671	32,553	41,840
5	123,783	238,304	269,660

(d)

4 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	0,022	0,022	0,022
2	0,044	0,044	0,041
3	0,978	1,754	2,349
4	0,567	0,783	N/A
5	61,629	N/A	N/A
Optimal			
1	0,023	0,022	0,022
2	0,065	0,077	0,068
3	1,753	3,215	5,079
4	1,207	1,761	N/A
5	126,689	N/A	N/A

the control synthesis cost is acceptable, however, with additional visibility windows (e.g., 4 visibility windows), additional pictures can be considered. Moreover, when we consider only few pictures ahead, in most of the cases (e.g. 1 or 2 pictures, with visibility windows > 1) the control synthesis overhead remains very low, thus compatible with the performances required by a short-horizon planner.

Plan Execution. As a final validation of the generated plan controllers we considered the time needed for plan execution comparing optimized and non-optimized controllers. In particular, the execution was simulated in UPPAAL-TIGA considering time average and variance of 20 runs. The uncontrollable events mentioned in the plan are randomly generated within their duration intervals.

The collected results are reported in Table 5 where we can observe a slight gain in time efficiency that seems negligible, in particular when compared with the generation cost associated with the optimization.

These results suggest that sub-optimal controllers provide a better trade-off between synthesis generation and plan execution.

Table 5: *Average durations and variances for plan controllers execution simulated by UPPAAL-TIGA.*

(a)

1 Comm. Window			
pic	0s flex	10s flex	20s flex
1	139±0	146±3	148±1
2	243±0	211±6	243±6
3	242±0	291±2	339±7
4	431±0	427±5	542±7
5	535±0	537±9	542±7
Optimal			
1	98±0	118±7	132±4
2	173±0	194±11	229±16
3	237±0	286±9	332±12
4	428±0	428±6	432±7
5	512±0	527±14	531±9

(b)

2 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	131±0	142±7	141±3
2	198±0	232±13	238±11
3	238±0	313±5	336±9
4	421±0	415±10	437±8
5	507±0	527±8	536±12
Optimal			
1	81±0	97±6	121±8
2	167±0	211±9	218±13
3	231±0	307±8	327±4
4	418±0	411±11	430±12
5	494±0	518±6	521±10

(c)

3 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	132±0	137±6	145±3
2	213±0	231±8	230±8
3	231±0	284±6	337±12
4	423±0	401±6	423±6
5	538±0	525±7	528±10
Optimal			
1	78±0	87±4	108±3
2	145±0	176±17	201±7
3	227±0	279±4	331±14
4	420±0	397±12	421±7
5	528±0	511±8	507±9

(d)

4 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	116±0	139±7	138±8
2	157±0	177±11	184±12
3	230±0	211±9	224±11
4	409±0	403±6	N/A
5	529±0	N/A	N/A
Optimal			
1	66±0	94±6	99±12
2	142±0	153±11	167±9
3	223±0	209±4	218±15
4	404±0	401±8	N/A
5	511±0	N/A	N/A

Conclusion

In this work, we have presented a formal method to automatically synthesize controllers for flexible temporal plans. Our approach allows to verify on-the-fly the dynamic controllability of a flexible temporal plan and, at the same time, generate its associated controller.

While flexible temporal plan execution is usually addressed using temporal constraint networks methods and algorithms to reduce the plan in a dispatchable form, here, an alternative novel technique based on the generation of a winning strategy in TGA has been investigated.

According to our approach, the plan execution problem can be solved completely as a side effect of dynamic controllability checking; hence, all the right plan execution decisions can be available before the plan execution with an acceptable overhead to the planning activity. The experimental results demonstrate the feasibility of the approach at work on a real world robotic scenario.

Acknowledgment. Cesta and Fratini are partially supported by EU under the ULISSE project (Contract FP7.218815). Cesta and Fratini are also partially supported by MIUR under the PRIN project 20089M932N (funds 2008) and by the European Space Agency (ESA) under the project GOAC (TRP/T313/006MM). Finzi is partially supported by EU under the AIRobots project (Contract FP7.248669). Orlandini is currently supported by a grant within "Accordo di Programma Quadro CNR-Regione Lombardia: Progetto 3".

Table 6: *Strategies dimension in terms of Kbytes and number of rules generated by UPPAAL-TIGA.*

(a)

1 Comm. Window			
pic	0s flex	10s flex	20s flex
1	47kb - 44	80kb - 56	133kb - 20
2	693k - 287	1447kb	2395kb - 593
3	1040kb - 352	2796kb - 717	8083kb - 1646
4	8348kb - 2100	12717kb - 2471	9925kb - 1714
5	25548kb - 5944	33743kb - 6174	39032kb - 6188
Optimal			
1	235kb - 139	299kb - 149	345kb - 160
2	2575kb - 797	3394kb - 870	3572kb - 913
3	10982kb - 2591	14290kb - 2860	16630kb - 3168
4	26522kb - 5514	34986kb - 6178	39284kb - 6430
5	101780kb - 11205	81791kb - 12110	79302kb - 11194

(b)

2 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	47kb - 44	80kb - 56	133kb - 78
2	141kb - 120	231kb -134	383kb - 153
3	852kb - 363	1037kb - 331	1800kb - 475
4	3424kb - 1087	6023kb - 1381	7226kb - 1436
5	10898kb - 2787	10662kb - 2149	21493kb - 3703
Optimal			
1	235kb - 139	299kb - 149	345kb - 160
2	383kb - 213	515kb - 229	592kb - 238
3	3771kb - 1146	4820kb - 1169	5656kb - 1291
4	11250kb - 2691	15066kb - 2984	17101kb - 1701
5	36023kb - 6309	38965kb - 7455	38977kb - 7546

(c)

3 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	20kb - 21	20kb - 18	20kb - 19
2	82kb - 79	73kb - 63	71kb - 58
3	305kb - 196	359kb - 195	541kb - 216
4	423kb - 229	747kb - 299	2415kb - 718
5	4888kb - 1459	5688kb - 1248	2939kb - 718
Optimal			
1	40kb - 39	42kb - 35	41kb - 36
2	110kb - 100	133kb - 106	131kb - 101
3	734kb - 347	875kb - 369	993kb - 378
4	3309kb - 991	4412kb - 1021	4606kb - 1108
5	9762kb - 2134	3297kb - 2893	3807kb - 3078

(d)

4 Comm. Windows			
pic	0s flex	10s flex	20s flex
1	20kb - 21	20kb - 18	20kb - 19
2	82kb - 79	73kb - 63	71kb - 58
3	378kb - 238	456kb - 223	697kb - 239
4	317kb - 243	452kb - 243	N/A
5	3685kb - 1191	N/A	N/A
Optimal			
1	40kb - 39	42kb - 35	41kb - 36
2	107kb - 99	134kb - 107	132kb - 102
3	774kb - 374	943kb - 401	1130kb - 403
4	491kb - 298	683kb - 336	N/A
5	9876kb - 2367	N/A	N/A

References

Abdedaim, Y.; Asarin, E.; Gallien, M.; Ingrand, F.; Lesire, C.; and Sighireanu, M. 2007. Planning Robust Temporal Plans: A Comparison Between CBTP and TGA Approaches. In *ICAPS-07. Proc. 17th Int. Conf. on Automated Planning and Scheduling*, 2–10.

Bensalem, S.; de Silva, L.; Gallien, M.; Ingrand, F.; and Yan, R. 2010. “Rock Solid” Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In *i-SAIRAS-10. Proceedings of the 10th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*.

Cassez, F.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, 66–80. Springer-Verlag.

Ceballos, A.; Bensalem, S.; Cesta, A.; De Silva, L.; Fratini, S.; Ingrand, F.; Ocon, J.; Orlandini, A.; Rajan; Rasconi, R.; and Van Winnendael, M. 2011. A goal-oriented autonomous controller for space exploration. In *11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 2011)*.

Cesta, A., and Fratini, S. 2008. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *PlanSIG-08. Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12*.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010. Analyzing Flexible Timeline-based Plans. In *ECAI-10. Proc. 19th European Conf. on Artificial Intelligence*, volume 215. IOS Press.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Gat, E. 1997. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press.

Goldman, R. P.; Musliner, D. J.; ; and Pelican, M. J. 2002. Exploiting implicit representations in timed automaton verification for controller synthesis. In *Proc. of HSCC-02*.

Lemai, S., and Ingrand, F. 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In *AAAI-04. Proc. 19th Nat. Conf. on Artificial Intelligence*.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the Synthesis of Discrete Controllers for Timed Systems. In *STACS*, LNCS, 229–242. Springer.

Morris, P. H., and Muscettola, N. 2005. Temporal Dynamic Controllability Revisited. In *AAAI-05. Proc. 20th Nat. Conf. on Artificial Intelligence*, 1193–1198.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic Control of Plans With Temporal Uncertainty. In *Proc. 17th Int. Joint Conf. on Artificial Intelligence*, 494–502.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Py, F.; Rajan, K.; and McGann, C. 2010. A Systematic Agent Framework for Situated Autonomous Systems. In *AAMAS-10. Proceedings of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*.

Shah, J., and Williams, B. C. 2008. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *ICAPS-08. Proc. 18th Int. Conf. Automated Planning and Scheduling*, 322–329.

Vidal, T., and Fargier, H. 1999. Handling Contingency in Temporal Constraint Networks: From Consistency To Controllabilities. *JETAI* 11(1):23–45.