Flexible Plan Verification: Feasibility Results

A. Cesta[1], A. Finzi[2], S. Fratini[1], A. Orlandini[3], and E. Tronci[4]

[1] ISTC-CNR, Via S.Martino della Battaglia 44, I-00185 Rome, Italy
amedeo.cesta,simone.fratini@istc.cnr.it

[2] DSF "Federico II" University, Via Cinthia, I-80126 Naples, Italy
finzi@na.infn.it

[3] DIA "Roma TRE" University, Via della Vasca Navale 79, I-00146 Rome, Italy
orlandin@dia.uniroma3.it

[4] DI "La Sapienza" University, Via Salaria 198, I-00198 Rome, Italy
enrico.tronci@di.uniroma1.it

Abstract

Timeline-based planning techniques have demonstrated wide application possibilities in heterogeneous domains. However, a problem for a wider diffusion of such technology still stems in the reduced community that has been studying formal properties of this planning approach. In this sense, we are currently studying the connection between plan generation and execution from the particular perspective of verifying a flexible plan before its actual execution. Nevertheless, a complexity issue should be considered. In fact, validation and verification processes usually exploit formal methods, such as model-checking, that present hard complexity. In this work, we explore how a model-checking verification tool, based on UPPAAL-TIGA, is suitable for verifying flexible temporal plans. In particular, we show the feasibility of our own approach by reporting some preliminary empirical results gathered in a real-world case study.

1 Introduction

Timeline-based planning has been shown very effective for applications in real-world domains – see [17, 11, 9, 19]. A problem for a wider diffusion of such technology stems in the limited community that has been studying with formal methodologies the properties of this planning approach. These authors have been investigating the interconnection between timeline-based planning and standard techniques for formal validation and verification (V&V) with the aim of building a rich environment for knowledge engineering [6] and exploring properties that concern temporal plans and their execution [7].

In this work, we consider the problem of flexible temporal plan verification. Flexible temporal plans only impose minimal temporal constraints among the planned activities, hence are able to on-line adapt to environmental changes. Such temporal flexibility makes plan verification complex and challenging (see [1] for a discussion). Here, we illustrate how this problem can be formalized and solved by using

Proceedings of the 16th International RCRA workshop (RCRA 2009):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Reggio Emilia, Italy, 12 December 2009

Timed Game Automata [14] and UPPAAL-TIGA [3]. In particular, to show the feasibility of the approach, we tackle the *controllability problem* [21, 16]. Such problem arises when a generated temporally flexible plan is to be executed by an *executive* system that manages controllable processes in the presence of exogenous events. In this scenario, the duration of the execution process is not completely under the control of the executive: the actions that are under the scope of the executive should be chosen so that they do not constrain uncontrollable events. Since [21] the controllability problem has been addressed through the temporal network that underlies a temporal plan, here we show how our general purpose verification method can be deployed to solve this relevant problem in flexible plan verification.

The controllability problem can be addressed in polynomial time under some simplifying assumptions [16]. [15] shows that this apparent low-order polynomial is misleading, because for many applications the input size may be very large in practical terms (pseudo-polynomial complexity). While a general purpose model-checker verification is PSPACE complete. Despite this theoretical complexity result, we found that UPPAAL-TIGA algorithm [3] yields very encouraging performance results in practice. Thus, the main contribution of the present paper is to show the feasibility of our approach presenting some experimental results on preliminary tests focusing on the analysis of the dependency of plan verification performance from the degree of *flexibility*. Results are gathered using a case study derived from a real-world space application scenario.

In the following we first present some related literature then the basic definition that ground our work, namely time-line based planning and Timed Game Automata. We then present the specific result of this work consisting in mapping temporal plans in timed automata and formally verifying the flexible plans. Hence we evaluate the whole approach. Some conclusions end the paper.

2 Related Works

[1] proposes a mapping from temporal constraint-based planning problems into UPPAAL-TIGA game-reachability problems providing a comparison of the two planning approaches. Here, the authors main concern was plan synthesis, while our current goal is flexible plan verification. The approach to problem modeling is similar to ours, however, in that work the flexibility issue remains open. Also in [12] we find a mapping from interval-based temporal relations models (i.e., Domain Description Language models from RAX-PS) to timed automata models of UPPAAL [13], however, again, flexible timeline verification is not addressed. Furthermore, [20] proposes a mapping from *Contingent Temporal Constraint Networks* (a generalization of STPUs) to *Timed Game Automata* which is analogous to the one exploited here. In this work, the use of a model checker is suggested only to obtain a more compact representation, but not to verify plan properties. In a PDDL framework, [10] tackle temporal plan verification, however, the authors do not address flexible temporal plans and other temporal features.

3 Timeline-Based Planning and Execution

Timeline-based planning is an approach to temporal planning [17] where the generated plans are represented by sets of timelines. Each timeline denotes the evolution of a particular feature in a dynamic system. A planning domain encodes the possible evolutions of the timelines whose time points have to satisfy temporal constraints, usually represented as Simple Temporal Problem (STP) restrictions.

Here, we assume that the timelines in a planning domain are incarnations of multi-valued *state variables* as in [17]. A state variable is characterized by a finite set of values describing its temporal evolutions, and by minimal and maximal duration for each value. 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 $[lb, ub]$). 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 $\mathcal{H}$. We suppose that adjacent intervals present different values. A timeline is said *completely specified* over the temporal horizon $\mathcal{H}$ when a sequence of non-overlapping valued intervals exists and its union is equal to $\mathcal{H}$. A timeline is said *time-flexible* when is completely specified and transition events are associated to temporal intervals (lower and upper bounds are given for them), instead of exact temporal occurrences. In other words, a time-flexible timeline represents a set of timelines, all sharing the same sequence of values. It is worth noting that not all the timelines in this set are valid (satisfies a-b-c). The process of *timeline extraction* from a time-flexible timeline is the process of computing (if exists) a valid and completely specified timeline from a given time-flexible timeline. In timeline-based planning, a *planning domain* is defined as a set of state variables $\{\mathcal{SV}_1, \ldots, \mathcal{SV}_n\}$ that cannot be considered as reciprocally decoupled. Then, a *domain theory* is defined as a set of additional relations, called *synchronizations*, that model the existing temporal constraints among state variables. A synchronization has the form $\langle \mathcal{TL}, v \rangle \longrightarrow \langle \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}, \{v'_1, \ldots, v'_{|\mathcal{TL}'|}\}, \mathcal{R} \rangle$ where: $\mathcal{TL}$ is the reference timeline; v is a value on $\mathcal{TL}$ which makes the synchronization applicable; $\{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}$ is a set of target timelines on which some values v'_j must hold; and $\mathcal{R}$ is a set of *relations* which bind temporal occurrence of the *reference* value v with temporal occurrences of the *target* values $v'_1, \ldots, v'_{|\mathcal{TL}'|}$. A *plan* is defined as a set of timelines $\{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ over the same interval for each state variable. A plan 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. A plan is *time flexible* if $\exists \mathcal{TL}_i \in \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ such that $\mathcal{TL}_i$ is time flexible.

At execution time, 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 these cases, the values for the state

variables that are under the executive scope should be chosen so that they do not constrain uncontrollable events. This *controllability problem* is defined, e.g. in [21] 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 have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) in [21] where basic formal notions are given for *dynamic* controllability (see also [15]). In the timeline-based framework, we introduce the same controllability concept defined on STNU as follows. Given a plan as a set of flexible timelines $\mathcal{PL} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$, we call *projection* the set of flexible timelines $\mathcal{PL}' = \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}$ derived from $\mathcal{PL}$ setting to a fixed value the temporal occurrence of each uncontrollable timepoint. Considering N as the set of controllable flexible timepoints in $\mathcal{PL}$, a *schedule* T is a mapping $T : N \rightarrow \mathbb{N}$ where $T(x)$ is called *time* of timepoint x. A *schedule* is *consistent* if all value durations and synchronizations are satisfied in $\mathcal{PL}$. The *history* of a timepoint x w.r.t. a schedule T, denoted by $T\{\prec x\}$, specifies the time of all uncontrollable timepoints that occur prior to x. An *execution strategy* S is a mapping $S : \mathcal{P} \rightarrow \mathcal{T}$ where $\mathcal{P}$ is the set of projections and $\mathcal{T}$ is the set of schedules. An execution strategy S is viable if $S(p)$ (denoted also S_p) is consistent for each projection p. Thus, a flexible plan $\mathcal{PL}$ is *dynamically controllable* if there exists a viable execution strategy S such that $S_{p1}\{\prec x\} = S_{p2}\{\prec x\} \Rightarrow S_{p1}(x) = S_{p2}(x)$ for each controllable timepoint x and projections $p1$ and $p2$.

4 Timed Game Automata

A fundamental concept in Timed Automata is time. First, we give the formal definition of clocks and relations that can be defined over them. We call *clock* a nonnegative, real-valued variable. Let X be a finite set of clocks. We denote with $C(X)$ the set of constraints Φ generated by the grammar: $\Phi ::= x \sim c \mid x - y \sim c \mid \Phi \wedge \Phi$, where $c \in Z$, $x, y \in X$, and $\sim \in \{<, \leq, \geq, >\}$. We denote by $B(X)$ the subset of $C(X)$ that uses only constraints of the form $x \sim c$.

Definition 1. *A* **Timed Automaton (TA)** *[2] 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, X *is a finite set of 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 *and each transition (*q*,* g*,* a*,* Y*,* q'*) is noted* $q \overset{g,a,Y}{\rightarrow} q'$.

A *valuation* of the variables in X is a mapping v from X to the set $R_{\geq 0}$ of nonnegative reals. We denote with $R^X_{\geq 0}$ the set of valuations on X and with $\vec{0}$ the valuation that assigns the value 0 to each clock. If $Y \subseteq X$ we denote with $v[Y]$ the valuation (on X) assigning the value 0 ($v(z)$) to any $z \in Y$ ($z \in (X - Y)$). For any $\delta \in R^{\geq 0}$ we denote with $(v + \delta)$ the valuation such that, for each $x \in X$, $(v + \delta)(x)$

= $v(x)$ + δ. Let $g \in C(X)$ and v be a valuation. We say that g satisfies v, notation $v \models g$ if constraint g evaluated on v returns true. A *state* of TA $\mathcal{A}$ is a pair (q, v) that $q \in Q$ and v is a valuation (on X). We denote with S the set of states of $\mathcal{A}$. An *admissible* state for $\mathcal{A}$ is a state (q, v) that $v \models \text{Inv}(q)$. A *discrete transition* for $\mathcal{A}$ is 5-tuple $(q, v) \xrightarrow{a} (q', v')$ where $(q, v), (q', v') \in S$, $a \in \text{Act}$ and there exists a transition $q \overset{g,a,Y}{\rightarrow} q' \in E$ that $v \models g$, $v' = v[Y]$ and $v' \models \text{Inv}(q')$. A *time transition* for $\mathcal{A}$ is 4-tuple $(q, v) \xrightarrow{\delta} (q, v')$ where $(q, v) \in S$, $(q, v') \in S$, $\delta \in R_{\geq 0}$, $v' = v + \delta$, $v \models \text{Inv}(q)$ and $v' \models \text{Inv}(q)$. A *run* of a TA $\mathcal{A}$ 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. A ***network* of TA (nTA)** is a finite set of TA evolving in parallel with a CSS style semantics for parallelism. Formally, let $\mathcal{F} = \{\mathcal{A}_i \mid i = 1, \ldots n\}$ be a finite set of automata with $\mathcal{A}_i = (Q_i, q_i^0, \text{Act}, X, \text{Inv}_i, E_i)$ for $i = 1, \ldots n$. Note that the automata in $\mathcal{F}$ have all the same set of actions and clocks and disjoint sets of locations. The *network* of $\mathcal{F}$ (notation $||\mathcal{F}$) is the TA $\mathcal{P} = (Q, q^0, \text{Act}, X, \text{Inv}, E)$ defined as follows. The set of locations Q of $\mathcal{P}$ is the Cartesian product of the locations of the automata in $\mathcal{F}$, that is $Q = Q_1 \times \ldots Q_n$. The *initial state* q^0 of $\mathcal{P}$ is $q^0 = (q_1^0, \ldots q_n^0)$. The *invariant* Inv for $\mathcal{P}$ is $\text{Inv}(q_1, \ldots q_n) = \text{Inv}_1(q_1) \wedge \ldots \text{Inv}_n(q_n)$. The *transition relation* E for $\mathcal{P}$ is the synchronous parallel of those of the automata in $\mathcal{F}$. That is, E consists of the set of 5-tuples (q, g, a, Y, q') satisfying the following conditions: 1. $q = (q_1, \ldots q_n)$, $q' = (q_1', \ldots q_n')$; 2. There are $i \leq j \in \{1, \ldots n\}$ such that for all $h \in \{1, \ldots n\}$, if $h \neq i, j$ then $q_h = q_h'$. Furthermore, if $i = j$ then action a occurs only in automaton $\mathcal{A}_i$ of $\mathcal{F}$. 3. Both automata $\mathcal{A}_i$ and $\mathcal{A}_j$ can make a transition with action a. That is, $q_i \overset{g_i,a,Y_i}{\rightarrow} q_i' \in E_i$, $q_j \overset{g_j,a,Y_j}{\rightarrow} q_j' \in E_j$, $g = g_i \wedge g_j$, $Y = Y_i \cup Y_j$.

Definition 2. *A* **Timed Game Automaton (TGA)** *is a TA where the set of actions* Act *is split in two disjoint sets:* Act_c *the set of controllable actions and* Act_u *the set of* uncontrollable *actions.*

The notions of network of TA, run, configuration are defined in a similar way for TGA.

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 starting from *Init* and executing f, $\mathcal{A}$ stays in *Safe* and reaches *Goal*. More precisely, 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 $f(\rho) = l$. A strategy f is *state-based* or *memory-less* whenever its result depends only on the last configuration of the run.

5 Building TGA from Timeline-based Planning Specifications

The main contribution of this work is the description of how flexible timeline-based plan verification can be performed by solving a Reachability Game using UPPAAL-TIGA. To this end, this section describes how a flexible timeline-based plan, state variables and domain theory can be formalized as an adequate nTGA. Timelines and state variables are mapped into TGA. In addition, a *Observer* TGA checks for both illegal values occurrences and synchronizations violations. Here, we distinguish between controllable and uncontrollable state variables/timelines to simplifying the formalization.

Given a flexible plan $\mathcal{P} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$, we define a TGA for each $\mathcal{TL}_i$. For each valued interval in the timeline (also called plan step), we consider a location in the automaton. An additional final location, labeled *goal*, is considered. We consider a unique *plan clock* c_p over all the timelines automata. Then, for each planned flexible timeline $\mathcal{TL}$, we define a Timed Game Automaton $\mathcal{A}_{\mathcal{TL}} = (Q_{\mathcal{TL}}, q_0, \text{Act}_{\mathcal{TL}}, X_{\mathcal{TL}}, \text{Inv}_{\mathcal{TL}}, E_{\mathcal{TL}})$ as follows. For each i-th valued interval in $\mathcal{TL}$, we consider l_i in $Q_{\mathcal{TL}}$, plus the final location l_{goal} (q_0 is l_0). For each allowed value $v \in \mathcal{SV}_i$, we consider an action a_v. If the related state variable is controllable (uncontrollable) we add a_v in $\text{Act}_{c\mathcal{TL}}$ ($\text{Act}_{u\mathcal{TL}}$). The overall plan clock c_p is considered in $X_{\mathcal{TL}}$. For each i-th valued interval in $\mathcal{TL}$ and the related value v_p associated with the flexible interval timepoint $[lb, ub]$, we define $\text{Inv}_{\mathcal{TL}}(l_i) := c_p \leq ub$ and we define a transition $e = q \overset{g,a,Y}{\rightarrow} q'$ in $E_{\mathcal{TL}}$, where $q = l_i$, $q' = l_{i+1}$, $g = c_p \geq lb$, $a = v_p!$, $Y = \emptyset$. Finally, we define a final transition $e = q \overset{g,a,Y}{\rightarrow} q'$ in $E_{\mathcal{TL}}$, where $q = l_{pl}$ (where pl is the plan length), $q' = l_{goal}$, $g = \emptyset$, $a = \emptyset$, $Y = \emptyset$. The set $Plan = \{\mathcal{A}_{\mathcal{TL}_1}, ..., \mathcal{A}_{\mathcal{TL}_n}\}$ represents the planned flexible timelines description as a nTGA.

For each state variable $\mathcal{SV}$, we have a one-to-one mapping into a Timed Game Automaton $\mathcal{A}_{SV} = (Q_{SV}, q_0, \text{Act}_{SV}, X_{SV}, \text{Inv}_{SV}, E_{SV})$. In fact, for each allowed value v in $\mathcal{V}$, we consider a location l_v in Q_{SV} (q_0 is set according to the initial value of the related planned timeline). Then, for each allowed value $v \in \mathcal{V}$, we consider an action a_v. If the state variable is controllable (uncontrollable), we consider a_v in Act_{cSV} (Act_{uSV}). A local clock c_{sv} is considered in X_{SV}. Finally, for each allowed value $v \in \mathcal{V}$ and both the associated $\mathcal{T}(v)$ = $\{vs_1, ..., vs_n\}$ and $\mathcal{D}(v)$ = $[l_b, u_b]$, we define $\text{Inv}_{SV}(v) := c_{sv} \leq u_b$ and we define a transition $e = q \overset{g,a,Y}{\rightarrow} q'$, where $q = l_v$, $q' = l_{vs_i}$ in E_{SV}, $g = c_{sv} \geq l_b$, $a = a_{vs_i}?$, $Y = \{c_{sv}\}$. The set $SV = \{\mathcal{A}_{SV_1}, ..., \mathcal{A}_{SV_n}\}$ represents the State Variables description. Note that the use of actions as transitions label implements the synchronization between state variables and planned timelines.

A last TGA is the *Observer* that monitors synchronizations and values over SV and $Plan$. Basically, two locations are considered to represent *correct* and *error* status. For each possible cause of error, an appropriate transition is defined, forcing the Observer to hold the error location. In this sense, we define a TGA

$\mathcal{A}_{Obs} = (Q_{Obs}, q_0, \text{Act}_{Obs}, X_{Obs}, \text{Inv}_{Obs}, E_{Obs})$ as follows. We consider $Q_{Obs} = \{l_{ok}, l_{err}\}$ (q_0 is l_{ok}), $\text{Act}_{uObs} = \{a_{fail}\}$, $X_{Obs} = \{c_p\}$. Inv_{Obs} is undefined. For each pair plan step and associated planned value (s_p, v_p) for each timeline $\mathcal{TL}$ and the related variable $\mathcal{SV}$, we define an uncontrollable transition $e = q \overset{g,l,r}{\rightarrow} q'$ in E_{Obs}, where $q = l_{ok}$, $q' = l_{err}$, $g = \mathcal{TL}_{s_p} \wedge \neg SV_{v_p}$, $l = a_{fail}$, $r = \emptyset$. Moreover, for each synchronization $\langle \mathcal{TL}, v \rangle \longrightarrow \langle \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}, \{v'_1, \ldots, v'_n\}, \mathcal{R} \rangle$, we define an uncontrollable transition $e = q \overset{g,a,Y}{\rightarrow} q'$ in E_{Obs} where $q = l_{ok}$, $q' = l_{err}$, $g = \neg\mathcal{R}(\mathcal{TL}_v, \mathcal{TL}'_{1v'_1}, \ldots, \mathcal{TL}'_{nv'_n})$, $a = a_{fail}$, $Y = \emptyset$.
The nTGA $\mathcal{PL}$ composed by the set of automata $PL = SV \cup Plan \cup \{\mathcal{A}_{Obs}\}$ encapsulates Flexible plan, State Variables and Domain Theory descriptions.

6 Verifying Time Flexible Plans

Given the nTGA $\mathcal{PL}$ defined above, we can define a Reachability Game that ensures, if successfully solved, plan validity. In particular, we define $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) by considering *Init* as the set of initial locations of each automaton in $\mathcal{PL}$, *Safe* = $\{l_{ok}\}$ and *Goal* as the set of goal locations of each $\mathcal{TL}_i$ in $\mathcal{PL}$. In [7], we demonstrate by construction that we obtain a one-to-one mapping between flexible behaviors, defined by $\mathcal{P}$, and automata behaviors, defined by *Plan* $\cup$ *SV*. While, the Observer holds the error location iff either an illegal value occurs or a synchronization is violated. Hence, $\mathcal{PL}$ adequately represents all and only the behaviors defined by the flexible plan $\mathcal{P}$.

To solve such a reachability game, we use UPPAAL-TIGA [3]. This tool extends UPPAAL [13] by providing a toolbox for the specification, simulation, and verification of real-time games. If there is no winning strategy, UPPAAL-TIGA gives a counter strategy for the opponent (environment) to make the controller lose. Given a nTGA, a set of goal states (*win*) and/or a set of bad states (*lose*), four types of winning conditions can be issued [3]. We ask UPPAAL-TIGA to solve the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) checking the formula $\Phi =$ *A [Safe U Goal]* in $\mathcal{PL}$. In fact, this formula means that along all the possible paths, $\mathcal{PL}$ stays in *Safe* states until *Goal* states are reached. In other words, winning the game corresponds to ask UPPAAL-TIGA to find a strategy that, for each possible evolution of uncontrollable state variables, ensures goals to be reached and errors to be avoided. Thus, verifying with UPPAAL-TIGA the above property implies validating the flexible temporal plan (see [7] for a formal account).

Moreover, we show the feasibility and effectiveness of our verification method by addressing the relevant issue of plan controllability. In fact, we can notice that each possible evolution of uncontrollable automata corresponds to a timeline projection p. Each strategy/solution for the RG corresponds to a schedule T. And a set of strategy represents an execution strategy S. Thus, the winning strategies produced by UPPAAL-TIGA constitute a viable execution strategy S for the flexible timelines. The use of forward algorithms [3] guarantees that S is such that

$S_{p1}\{\prec x\} = S_{p2}\{\prec x\} \Rightarrow S_{p1}(x) = S_{p2}(x)$ for each controllable timepoint x and projections $p1$ and $p2$. That is, the flexible plan is dynamically controllable.

7 Case Study and Preliminary Experiments

In this section, we present the application of our method in a specific case study. In our recent work we have considered variants of a real application case studies [8, 6]. The same experience has been used here to derive a general planning problem. Basically, a remote space agent is to be controlled in order to accomplish some required tasks (science, communication and maintenance activities). Tasks have to be temporally synchronized with exogenous events that occur independently from the agent control.

We represent the domain problem with two different types of state variables: *Controllable State Variables*, which define the search space of the problem, and whose timelines ultimately represent the solution to the problem; *Uncontrollable State Variables*, representing values imposed over time which can only be observed. Modeling the agent activities, we use a single controllable state variable which specifies the temporal occurrence of science and maintenance operations as well as the agent's ability to communicate. Additional values are considered in order to represent *earth pointing* and *slewing manoeuvres* (resp. *Earth* and *Slew* modalities). The values that can be taken by this state variable, their durations and the allowed transitions among them, are detailed in Figure 1.

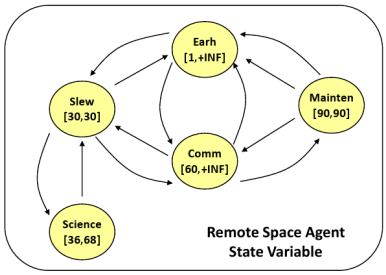

Figure 1: Value transitions for the a main state variable describing the Remote Space Agent temporal behavior.

In addition, we instantiate two uncontrollable state variables to represent contingent events such as orbit events and communication opportunity windows. One state variable maintains the temporal occurrences of pericentres and apocentres. We are supposing the remote agent is operative around a target planet. Pericentre is the orbital closest to the target planet while apocentre is the orbital far away from the planet. ("PERI" and "APO" values on the timeline in Figure 2, top) of the agent's orbit (they are fixed in time), while the other state variable maintains the visibility of ground stations (Ground Station Availability timeline in Figure 2, bottom). This state variable has as allowed values $\{$*Available, Unavailable*$\}$.

Any valid plan needs synchronizations among the agent timeline (Figure 2, middle) and the uncontrollable timelines (represented as dotted arrows in Figure 2): science operations must occur during Pericentres, (meaning that a *Science* value must start and end during a *Peri* value); maintenance operations must occur in the same time interval as Apocentres (meaning that a *Maint* value must start and end

exactly when the *Apo* value starts and ends); communications must occur during ground station visibility windows. (meaning that a *Comm* value must start and end during an *Available* value). In addition to those synchronization constraints, the operative mode timeline must respect transition constraints among values and durations for each value specified by the domain (see again Fig. 2).

7.1 Using UPPAAL-TIGA

We now show how planning domains can be encoded in the specification language of UPPAAL-TIGA. This requires defining a suitable set of automata and clocks. Automata are associated with multi-valued state variables while clocks are necessary to represent time progress.

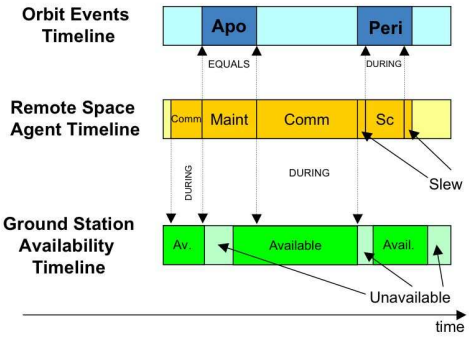

Figure 2: Timeline synchronizations in a plan.

For each state variable (and hence for each timeline) we have a *state variable timed automaton* whose modes correspond to possible state variable values, while transitions represent changes of values. State variable definition includes temporal constraints specified by means of: value durations constraints (in terms of $[min, max]$); sequencing constraints between values expressed through Allen's temporal relations.

Durations constraints (e.g., Science activity duration in $[2160, 4080]$) are encoded as both clock mode invariants and guards on the related outgoing transitions. While sequencing constraints (e.g., Science *meets* Slew) are encoded defining appropriate outgoing transitions. In Figure 3 we report the complete UPPAAL-TIGA module declaration for the agent state variable.

Plan verification requires an input model that encodes also the generated plan. Since a generated plan provides a set of value activations (associated with time points) (planned timeline) for each state variable, a plan describes the sequence of values the state variables are to assume in a given time frame. To represent flexible plans, we consider an additional general *plan clock* and we introduce an automaton for each planned behavior. This automaton has a number of modes that equals the length of the plan: for each activation/decision available in the plan we introduce a mode while a final *goal* mode represents plan completion. An invariant is considered to model maximum staying duration. Transitions between modes represent plan steps, from initial value to the last one. For each transition, we introduce a guard that enables transition at the minimum staying duration.

In order to consider both controllable and uncontrollable state variables, we introduce uncontrollable TGA transitions for uncontrollable components.

```
process REMOTE_AGT() {
state
    Earth, Earth_Comm,
    Science {clockREMOTE_AGT <= 4080},
    Maintenance {clockREMOTE_AGT <= 5400},
    Slew {clockREMOTE_AGT <= 1800};
init Earth;
trans
    Earth -> Slew { guard clockREMOTE_AGT >= 1;
                              sync pulse_Slew?;  },
    Earth -> Maintenance { guard clockREMOTE_AGT >= 1;
                              sync pulse_Maintenance?;
                              assign clockREMOTE_AGT := 0;},
    Earth -> Earth_Comm { guard clockREMOTE_AGT >= 1;
                              sync pulse_Earth_Comm?;
                              assign clockREMOTE_AGT := 0;},
    Earth_Comm -> Earth { guard clockREMOTE_AGT >= 3600;
                              sync pulse_Earth?;
                              assign clockREMOTE_AGT := 0;},
    Earth_Comm -> Maintenance { guard clockREMOTE_AGT >= 3600;
                              sync pulse_Maintenance?;
                              assign clockREMOTE_AGT := 0;},
    Earth_Comm -> Slew { guard clockREMOTE_AGT >= 3600;
                              sync pulse_Slew?;
                              assign clockREMOTE_AGT := 0;},
    Science -> Slew { guard clockREMOTE_AGT >= 2160;
                              sync pulse_Slew?;
                              assign clockREMOTE_AGT := 0;},
    Maintenance -> Earth { guard clockREMOTE_AGT >= 5400;
                              sync pulse_Earth?;
                              assign clockREMOTE_AGT := 0;},
    Maintenance -> Earth_Comm { guard clockREMOTE_AGT >= 5400;
                              sync pulse_Earth_Comm?;
                              assign clockREMOTE_AGT := 0;},
    Slew -> Earth { guard clockREMOTE_AGT >= 1800;
                              sync pulse_Earth?;
                              assign clockREMOTE_AGT := 0;},
    Slew -> Earth_Comm { guard clockREMOTE_AGT >= 1800;
                              sync pulse_Earth_Comm?;
                              assign clockREMOTE_AGT := 0;},
    Slew -> Science { guard clockREMOTE_AGT >= 1800;
                              sync pulse_Science?;
                              assign clockREMOTE_AGT := 0;};
}
```

Figure 3: Module definition for the Remote Space Agent.

In Figure 4, two encoded *plan automata* are depicted: a) a flexible plan for the remote agent that is to be verified; b) a behavior of the ground station availability state variable. Note that synchronization channels are exploited to relate planned values to state variables automaton. For instance, the second transition in Figure 4a synchronizes with related transition defined in Figure 3 between Slew and Science modes.

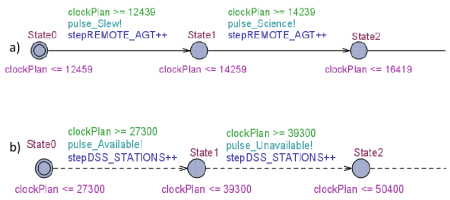

Figure 4: TIGA models for timelines: a) controllable state variable; b) uncontrollable state variable.

In addition, we introduce another automaton: the *observer automaton*. It is to check the consistency of temporal constraints defined both on and among different timelines, i.e., to check sequencing and synchronizations constraints. Synchronization constraints among different timelines are expressed in terms of general temporal relations on values.

```
process monitor() {
state OK, ERR;
init OK;
trans
    OK -u-> ERR { guard (stepREMOTE_AGT == 0)
                        and not (REMOTE_AGTEarth);  },
    OK -u-> ERR { guard (stepREMOTE_AGT == 1)
                        and not (REMOTE_AGTSlew);  },
    ...
    OK -u-> ERR { guard ((REMOTE_AGTEarth_Comm)
                        and not (STATIONSAvailable));  },
    OK -u-> ERR { guard ((REMOTE_AGTMaintenance)
                        and not (ORBIT_EVENTSApocentre));  },
    OK -u-> ERR { guard ((REMOTE_AGTScience)
                        and not (ORBIT_EVENTSPericentre));  },
    ERR -u-> ERR {  };
}
```

Figure 5: Partial monitor module definition. Note that Monitor is uncontrollable.

Given the above input model, we ask UPPAAL-TIGA to verify the following formula: *control: A [not monitor.ERR U plan.Goal]*. This formula means that for each possible evolution of uncontrollable components, the goal must be reached while monitor errors must be avoided. If verified, UPPAAL-TIGA returns a control execution strategy that, if respected, guarantees to reach planning goal in all possible world evolutions. Thus, verifying the above property implies validating the flexible temporal plan.

Since the input model incorporates all domain temporal constraints, the UPPAAL-TIGA verification algorithms guarantee that all time points in the strategy only depend on occurrences of past events. Such a feature constitutes the condition of dynamic controllability for a flexible temporal plan. So, verifying the formula not only guarantees plan validity, but it also ensures dynamic controllability.

7.2 Empirical Results

It has been demonstrated [18] that model checking complexity is PSPACE complete in the size of the system state space. Unfortunately, the number of system states is exponential in the size of the program defining the system (our input). For example, a 32 bit integer variable yields 2^{32} states. As a result, even a polynomial verification algorithm (the best we can hope for, by [18]) will take an exponential amount of memory (*state explosion*). Fortunately, in most cases real world verification problems are not hard instances. Indeed, the success of model checking rests on this fact and on the availability of verification algorithms that can successfully exploit the specific structure of verification problems stemming from real world applications.

Along the same line of thinking, our goal in this section is not to assess the worst case complexity of our approach, (which is known to be exponential in the input size), but rather to evaluate its effectiveness on meaningful real world examples.

In order to show the feasibility of our approach, we present experimental results on preliminary tests focusing on the analysis of the dependency of plan verification

performance from the degree of *flexibility*.

We generate a flexible plan by introducing flexibility into a completely instantiated plan. This is done by replacing a time point $t = \tau$ in the instantiated plan with a time interval $t \in [\tau - \Delta, \tau + \Delta]$ in the flexible plan. The main parameters we consider are: the number Φ of time points that are replaced with time intervals and the width (*duration*) Δ of such intervals.

Here, we perform two kind of experiments. First, by keeping Δ constant ($\Delta = 10$), we study how plan verification time depends on the plan size (i.e., the number of plan time points) and on the number of flexible time points Φ. Second, by keeping constant the plan size (to 35 time points), we study how plan verification time depends on the number of flexible time points Φ and on the duration Δ.

We run our experiments on a Linux workstation endowed with a 64-bit AMD Athlon CPU (3.5GHz) and 2GB RAM. Given Φ and Δ, an experiment consists in choosing at random Φ plan time points, replacing such chosen time points with time intervals of duration Δ, running the UPPAAL-TIGA verifier and, finally, measuring the verification time. For each configuration, we repeat our experiment 5 times and compute the mean value (in msecs.) and variance ($\pm var$) for the verification time.

We note that not all the experiments relative to given values for Φ and Δ yield a satisfiable flexible temporal plan. In fact, since the plan is only flexible at certain time points, the degrees of freedom may not suffice to recover from previously delayed (or anticipated) actions. Of course, this is particularly the case when Φ is small with respect to the plan size. Accordingly, our verification times refer to passing (i.e., the given flexible temporal plan is dynamically controllable) as well as failing (i.e., the given flexible temporal plan is not dynamically controllable) experiments.

Table 1 shows our results for the first kind of experiments. From this figure we see that the verification tool shows homogeneous performances over all the configurations.

Table 2 shows our results for the second kind of experiments. From this figure we see that the verification tool handles well flexible plan with higher and higher degrees of flexibility both in terms of Φ and Δ.

Table 1: Experimental results collected varying plan length and the number of flexible time points(Timings in msecs.)

Φ \ plan size	10	20	35
3	35.6 $\pm$0.8	36.6$\pm$1.7	37.4 $\pm$0.5
6	35.2 $\pm$0.4	36 $\pm$0	37.4 $\pm$0.5
9	36 $\pm$1.8	36.2 $\pm$0.4	39.2 $\pm$1.9
12	34.8 $\pm$0.4	36.4 $\pm$0.5	37.8 $\pm$0.4
15	35 $\pm$0	36.2 $\pm$0.4	43.6 $\pm$10.2
18	35 $\pm$0	40 $\pm$8	39 $\pm$0

Table 2: Experimental results collected with a fixed plan length (Timing in msecs.).

Φ \ Δ	1	5	10	15	20
3	40±6	37.4±0.5	37.8±0.4	51±7.8	37.8±1
6	38.4±0.5	38.6±1.2	38±0	44.4±8.5	38.2±0.4
9	38.4±0.5	38±0	39.2±1.9	39±0	38.8±0.4
12	52.4±10.3	38.8±0.4	38.4±0.5	39±0	39.4±0.5
15	39.2±0.4	52±13	39.2±0.4	39.2±0.4	39.8±0.4
18	39.6±0.5	39.6±0.8	40.4±1.5	48.8±9.1	40±0.6

8 Conclusion

This paper introduces a method to represent and verify flexible plans using TGA and UPPAAL-TIGA. In particular, it describes the verification method, detailing the formal representation and the modeling methodology. To show the feasibility and the effectiveness of the approach we have considered the relevant problem of dynamic controllability checking.

As well known, [18], in the worst case model checking requires an amount of memory exponential in the input size (*state explosion*). Fortunately, not all verification problems fall in the worst case category. Indeed, depending on the problem at hand effective heuristics have been devised to carry out verification with a reasonable amount of memory (and time). For example, UPPAAL-TIGA yields very encouraging performance results on some interesting classes of verification problems [4]. In much the same vein, the results presented here show that UPPAAL-TIGA allows effective verification of interesting flexible temporal plan verification problems.

Thus, model-checking in UPPAAL-TIGA on the one hand provides a useful independent verification tool for flexible timelines, on the other hand permits plan verification of the flexible plans produced by a black-box planner. Moreover, it produces results that can be further exploited as follows. First, from a valid flexible plan we can extract a strategy that can be used to safely execute the given plan. Second, an invalid plan can be analyzed and information can be obtained by the tool to diagnose the problem and get hints on how to obtain a valid plan.

Acknowledgments

Cesta, Fratini, Orlandini and Tronci are partially supported by the EU project ULISSE (Call “SPA.2007.2.1.01 Space Science”. Contract FP7.218815). Cesta and Fratini are also partially supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). Aspects from this paper are synthetically presented in [5].

References

[1] Y. Abdedaim, E. Asarin, M. Gallien, F. Ingrand, C. Lesire, and M. Sighireanu. Planning Robust Temporal Plans: A Comparison Between CBTP and TGA Approaches. In *ICAPS-07. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 2–10, 2007.

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

[3] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, pages 121–125. Springer, 2007.

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

[5] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible Timeline-Based Plan Verification. In *KI 2009*, volume 5803 of *LNAI*, 2009.

[6] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review (To appear)*, 2009.

[7] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Verifying flexible timeline-based plans. In *VVPS-09. Workshop on Verification and Validation of Planning and Scheduling Systems at ICAPS, Thessaloniki, Greece.*, September 2009.

[8] A. Cesta, S. Fratini, A. Oddi, and F. Pecora. APSI Case#1: Pre-planning Science Operations in MARS EXPRESS. In *i-SAIRAS-08. Proceedings of the 9^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*. JPL, Pasadena, CA, 2008.

[9] J. Frank and A. Jonsson. Constraint Based Attribute and Interval Planning. *Journal of Constraints*, 8(4):339–364, 2003.

[10] R. Howey and D. Long. VAL's Progress: The Automatic Validation Tool for PDDL2.1 Used in the International Planning Competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*, pages 28–37, Trento, Italy, June 2003.

[11] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 177–186, 2000.

[12] L. Khatib, N. Muscettola, and K. Havelund. Mapping Temporal Planning Constraints into Timed Automata. In *TIME-01. The Eigth Int. Symposium on Temporal Representation and Reasoning*, pages 21–27, 2001.

[13] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

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

[15] P. H. Morris and N. Muscettola. Temporal Dynamic Controllability Revisited. In *Proc. of AAAI 2005*, pages 1193–1198, 2005.

[16] P. H. Morris, N. Muscettola, and T. Vidal. Dynamic Control of Plans With Temporal Uncertainty. In *Proc. of IJCAI 2001*, pages 494–502, 2001.

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

[18] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[19] D. Smith, J. Frank, and A. Jonsson. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[20] T. Vidal. Controllability Characterization and Checking in Contingent Temporal Constraint Networks. In *Proceedings of KR-00*, 2000.

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