Fundamenta Informaticae 107 (2011) 111–137
DOI 10.3233/FI-2011-397
IOS Press

Flexible Plan Verification: Feasibility Results

Amedeo Cesta, Simone Fratini, Andrea Orlandini[*†]
Consiglio Nazionale delle Ricerche, Istituto di Scienze e Tecnologie della Cognizione
Via S.Martino della Battaglia 44, I-00185 Rome, Italy
{andrea.orlandini, amedeo.cesta, simone.fratini}@istc.cnr.it

Alberto Finzi[‡]
Dipartimento di Scienze Fisiche, Università di Napoli "Federico II"
Via Cinthia, I-80126 Naples, Italy
finzi@na.infn.it

Enrico Tronci[§]
Dipartimento di Informatica, Università di Roma "La Sapienza"
Via Salaria 198, I-00198 Rome, Italy
enrico.tronci@di.uniroma1.it

Abstract. Timeline-based planning techniques have demonstrated wide application possibilities in heterogeneous real world domains. For a wider diffusion of this technology, a more thorough investigation of the connections with formal methods is needed. This paper is part of a research program aimed at studying the interconnections between timeline-based planning and standard techniques for formal validation and verification (V&V). In this line, an open issue consists of studying the link between plan generation and plan execution from the particular perspective of verifying temporal plans before their actual execution. The present work addresses the problem of verifying flexible temporal plans, i.e., those plans usually produced by least-commitment temporal planners. Such plans only impose minimal temporal constraints among the planned activities, hence are able to adapt to on-line environmental changes by trading some of the retained flexibility.

[*]Authors are partially supported by EU under the ULISSE project (Contract FP7.218815), and MIUR under the PRIN project 20089M932N (funds 2008). Cesta and Fratini have been partially supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI).
[†]Address for correspondence: Via S.Martino della Battaglia 44, I-00185 Rome, Italy
[‡]Author is partially supported by EU under the DEXMART project (Contract FP7.216239)
[§]Author is partially supported by EU under the ULISSE project (Contract FP7.218815)

This work shows how a model-checking verification tool based on Timed Game Automata (TGA) can be used to verify such plans. In particular, flexible plan verification is cast as a model-checking problem on those automata. The effectiveness of the proposed approach is shown by presenting a detailed experimental analysis with a real world domain which is used as a flexible benchmark.

Keywords: Timeline-based Planning, Validation and Verification, Model Checking.

1. Introduction

Timeline-based planning has shown to be very effective for applications in real-world domains – see [24, 16, 27, 13, 5, 11]. Notwithstanding those successes, the dissemination of the technology is rather poor mostly because a limited amount of work has been dedicated to clarify the theoretical underpinnings and the formal properties of this approach to planning. These authors are investigating one of the missing aspects: the interconnection between timeline-based planning and standard techniques for formal validation and verification (V&V). Objectives of this research are (a) the synthesis of richer environments for knowledge engineering [7] and (b) the exploration of properties connecting temporal plans to their execution [8]. This specific work addresses the problem of flexible temporal plans verification and reinforce the preliminary results published in [8] and [6].

Flexible temporal plans identify an envelope of potential plans which represent feasible solutions to a given problem. Such plans only impose minimal temporal constraints among the planned activities, hence they are able to adapt to environmental changes by trading some of their flexibility during execution. It is worth underscoring how the verification of flexible plans is considered an open issue in the planning and scheduling literature (see [7, 1] for a discussion). Here, this verification problem is tackled with an approach based on a Timed Game Automata [20] formalization, exploiting UPPAAL-TIGA [3] as model checker.

In showing the feasibility of the approach, we tackle the *controllability problem* as defined in [29, 23]. This problem arises when a generated time flexible plan has to be executed by an *executive* system that manages controllable processes in the presence of exogenous events. In such a scenario, the duration of the execution processes is not completely under the control of the executive: the actions that are under the scope of the executive should therefore be chosen so that they do not constrain *uncontrollable* events. Since [29], the controllability problem is addressed through the temporal network that underlies the time flexible plan, here we show how our general purpose verification method can be used to solve this relevant problem in timeline-based planning.

The controllability problem can be addressed in polynomial time under some simplifying assumptions [23]. In [22] this apparent low-order polynomial complexity is shown to be misleading because for many real applications the input size may be very large (pseudo-polynomial complexity); on the other hand, a general purpose model-checker verification is PSPACE complete. Despite this theoretical complexity result, this paper discusses how the UPPAAL-TIGA algorithm can yield very encouraging performance results in practice. Thus, the main contribution of the present work is to show the feasibility of the approach, by presenting some experimental results that illustrate the plan verification performances, as the degree of plan *flexibility* and *controllability* varies. The experimental analysis is based on a proposed benchmark case study derived from a real world space application scenario. Such a case study is both realistic and rich enough to allow the implementation of a wide range of different experiment settings.

Related Works. A selected number of works share some methodology with our approach and somehow paved the way for its contribution. In [1], the authors propose a mapping from temporal constraint-based planning problems into UPPAAL-TIGA game-reachability problems providing a comparison of the two planning approaches. In that work, the main concern is plan synthesis, while our main focus is flexible plan verification. In [1], the approach to problem modeling is similar to ours; however, the temporal flexibility issue remains open. In [17] a mapping is proposed from interval-based temporal relations models (i.e., Domain Description Language models from RAX-PS [16]) into timed automata models of UPPAAL [19], again, flexible timeline verification is not addressed. Furthermore, [28] proposes a mapping from *Contingent Temporal Constraint Networks* (a generalization of STPUs) to *Timed Game Automata* which is analogous to the one exploited here. It is worth noting how in [28] the use of a model checker is suggested only to obtain a more compact representation, as opposed to verifying plan properties. In [15], the authors tackle temporal plan verification within a PDDL framework; however, they do not address flexible temporal plans and temporal constraints.

Plan of the Paper. In the following, the basic definitions grounding our work are first presented, namely, (a) Timeline-based Planning (Section 2) and (b) Timed Game Automata (Section 3). Then the specific result of the paper is introduced by (c) describing the automated mapping of a plan in timeline specification into TGAs (Section 4), and (d) introducing the basic verification procedure (Section 5). Furthermore, the efficacy of the approach is tested by (e) introducing the challenging benchmark from a space domain (Section 6) and (f) performing a number of diversified experiments (Section 7). Some conclusions end the paper.

2. Timeline-Based Planning and Execution

Timeline-based planning is an approach to temporal planning, proposed in the 90's [25, 24], which has been applied in the solution of several real world problems – e.g., [16, 5, 11]. The approach pursues a general idea that planning and scheduling consist in the synthesis of *desired temporal behaviors* for complex physical systems. In this respect, the set of domain features under control are modeled as a set of temporal functions whose values have to be planned over a time horizon. Such functions are synthesized during problem solving by posting planning decisions. The evolution of a single temporal feature over a time horizon is called the *timeline* of that feature.

In the rest of this paper, multi-valued *state variables* will represent time varying features, as defined in [24, 10]. As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal evolutions of timelines. For the state variables, such causal laws are encoded in a *domain specification* which determines the operational constraints of a given domain. In this context, the task of a planner is to find a sequence of control decisions that bring the variables into a final set of desired evolutions always satisfying the domain specification.

We want to represent temporal features as state-variables that have a finite set of possible values over temporal intervals. The temporal evolutions of such features are sequences of operational states – i.e., stepwise constant functions of time. Operational 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 $[lb, ub]$). $\mathcal{T}$ and $\mathcal{D}$ specify the operational constraints on the values in $\mathcal{V}$.

It is worth introducing here a distinction between state variable types that are used for modeling domains. We call *Planned State Variables* those whose activities are under the control of the planning agent (planned by the agent), and *External State Variables* those whose values can only be observed over time.[1]

Simple example. We introduce here a simple domain to exemplify the introduced concepts. Our example domain is composed of two relevant features: a robotic platform and a camera carried by the platform. The robot is a basic mobile platform with two operational states represented by the values: (a) `AT`($?object$) to model the situation where the platform is close to a potential target, and (b) `Move`($?object_1$,$?object_2$) to model the status when the platform is moving from a target to another target ($?object$ represents a parameter, whose values belong to a finite set of symbols, used to compactly represent the allowed values for a given state variable). The operational constraints state that the platform moves among the targets, thus allowing the following transitions between operational states: `AT`($?object_1$) $\rightarrow$ `Move`($?object_1$,$?object_2$) $\rightarrow$ `AT`($?object_2$). There is no duration constraint for the status `AT`($?object_1$) (it can be represented by the interval $[1, \infty]$), while we suppose a duration constraint $[d_{min}, d_{max}]$ for the status `Move`($?object_1$,$?object_2$) (it is reasonable that the time needed to move between two objects is bounded and dependent from the distance between the two objects). The camera can assume the following values: (c) `Idle`(), (d) `WarmUp`() and (e) `TakePicture`($?object$). The camera operational constraints require that the camera has to be warmed up before taking a picture. Thus, the following transitions represent a valid temporal evolution for the subsystem: `Idle`() $\rightarrow$ `WarmUp`() $\rightarrow$ `TakePicture`($?object$) $\rightarrow$ `Idle`(). Besides that, there exist duration constraints for the operational states `WarmUp`() and `TakePicture`($?object$). The warming up status has a fixed duration $[d_{wu}, d_{wu}]$, constant and known in advance, while the taking picture status has a minimal and maximal duration $[d_{min}, d_{max}]$: in fact, this is only a predicted duration for the time needed to take a picture and store the data. Figure 1-left shows the values of the two state variables and their legal transitions.

In this type of planning, a *planning domain*, or *domain theory*, is defined as a set of state variables $\{\mathcal{SV}_1, \ldots, \mathcal{SV}_n\}$. Generally, they are not mutually decoupled; rather, a set of additional relations exist, called *synchronizations*, that model the existing temporal and causal constraints among the values taken by different state variable timelines (i.e., patterns of legal occurrences of the operational states across the timelines). More formally, a synchronization has the form

$$\langle \mathcal{SV}, v \rangle \longrightarrow \langle \{\langle \mathcal{SV}'_1, v'_1 \rangle \ldots, \langle \mathcal{SV}'_n, v'_n \rangle\}, \mathcal{R} \rangle$$

where: $\mathcal{SV}$ is the reference state variable; v is a value on $\mathcal{SV}$ which makes the synchronization applicable; $\{\langle \mathcal{SV}'_1, v'_1 \rangle \ldots, \langle \mathcal{SV}'_n, v'_n \rangle\}$ is a set of target state variables 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

[1] In previous papers of ours (e.g., [7, 5]) we have used *Controllable* for *Planned* and *Uncontrollable* for *External* state variables. We have changed the terminology here to avoid the semantic clash with the controllability of single events which is a relevant concept in this paper.

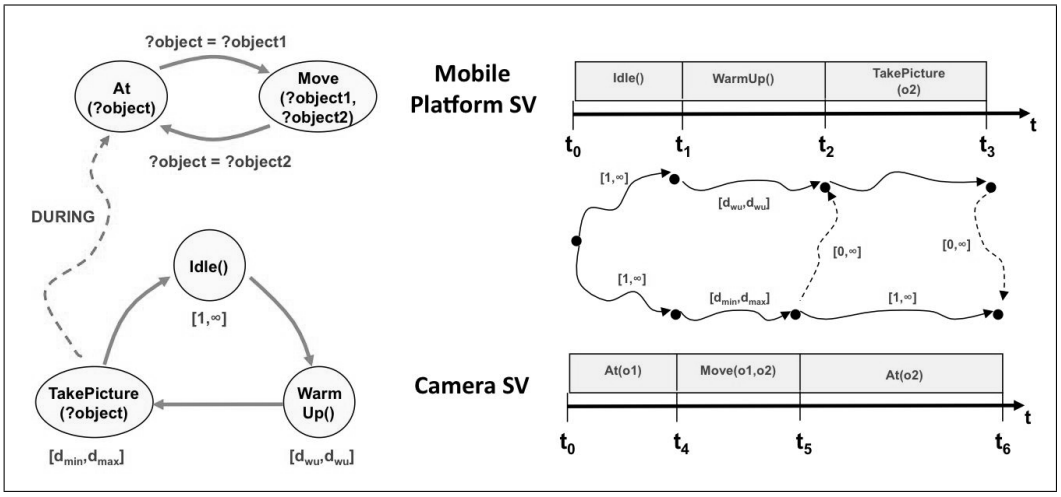

Figure 1: The simple timeline-based planning domain.

of the *target* values.[2] For instance, in Figure 1-left there is a dashed edge labeled with the temporal constraint DURING between the operational status TakePicture(?*object*) of the camera equipment and the operational status AT(?*object*) of the motion equipment. This synchronization models that in order to take a picture of a given target object, the robot must be close to the object. In terms of timeline-based planning, this means that the operational status TakePicture(?*object*) on the timeline that describes the behavior of the camera equipment must hold during the operational status AT(?*object*) of the timeline that describes the behavior of the motion equipment. In Figure 1, the temporal constraint DURING holds between the intervals $[t_2, t_3)$ and $[t_5, t_6)$ as a precedence constraint between the pairs t_2, t_5 and t_3, t_6.

Timeline based planning. The temporal evolutions of a state variable will be described by means of a *timeline*, 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; the latter are bounded by the planning process, i.e., they are assigned lower and upper bounds instead of being exactly specified. As in case of a least-commitment solving approach, we refer to the timeline as *time flexible* and we refer to the plan resulting from a set of flexible timelines as a *flexible plan*. Figure 1-right shows two timelines for the state variables previously introduced, along with the underlying temporal network that describes the constraints that hold among the flexible transition points in the timelines. In the temporal network, the nodes represent *transition points* and the edges represent *distance constraints* characterized by an upper and lower bound.

More formally, given a state variable $\langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$, a time flexible timeline is defined over a temporal interval $\mathcal{H} = [t_O, t_H)$, called *Planning Horizon*, by a tuple $\langle \mathcal{T}_{TL}, \mathcal{O}_{TL}, \mathcal{V}_{TL}, \mathcal{D}_{TL} \rangle$ where: (d) $\mathcal{T}_{TL} = \{tp_1, \ldots, tp_n\}$ is a finite set of *timepoints* such that $tp_i \in \mathcal{H}$; (e) $\mathcal{O}_{TL} : \mathcal{T}_{TL} \rightarrow \mathbb{N} \times \mathbb{N}$ is the *transition*

[2] A temporal relation $\mathcal{R}$ among two values A and B prescribes both qualitative and quantitative temporal constraints (e.g., A DURING B, or A OVERLAPS [min,max] B) thus using a combination of qualitative Allen's interval algebra and quantitative temporal expressions [10]. A temporal relation $\mathcal{R}$ stems for a set of temporal precedence constraints between the starting and ending points of occurrences of intervals where values A and B are assumed.

point occurrence function, i.e., a function that specifies the allowed lower and upper bounds of the transition points in $\mathcal{T}_{TL}$ (as an interval $[lb, ub]$);[3] (f) $\mathcal{V}_{TL} : \mathcal{T}_{TL} \rightarrow \mathcal{V}$ is the *value taken* function, i.e., a function that specifies the value taken by the timeline up to the following transition points;[4] (g) $\mathcal{D}_{TL} : \mathcal{T}_{TL} \rightarrow \mathbb{N} \times \mathbb{N}$ is the *transition distance* function, i.e., a function that specifies the allowed distance between two successive transition points (as an interval $[lb, ub]$).[5] A time flexible timeline represents a set of temporal evolutions of a state variable, all sharing the same sequence of values but with different temporal occurrences of the transition points. It is worth pointing out that not all the assignments of values to transition points specified by (f) are legal with respect to the state variable constraints (b) and not all the temporal assignments of the transition points specified by (e) are legal with respect to the state variable constraints (c) and the timeline constraints (g). Notice that *planning goals* are expressed as desired values over temporal intervals and the task of the planner is to build timelines that describe *valid* sequences of values that achieve those goal values – where valid means that the plan completely satisfies the domain theory.

Plan execution. During plan execution the plan is under the supervision of an executive program that forces value transitions over timelines. A well known problem with execution is that not all the value transitions are under the responsibility of the executive; rather, some of the events are under the 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 out of its control. In such 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 is the *controllability problem* which was defined for example in [29] where *contingent* and *executable* processes are distinguished. The contingent processes are not controllable, and hence are characterized by uncertain durations. Instead, the executable processes are started and ended by the executive system. In particular, controllability issues underlying a plan representation have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) representation in [29] where basic formal notions are given for *dynamic* controllability (see also [22]).

In [6] we extended the controllability concepts defined on STPUs to a timeline-based representation. The main steps of such an extension are summarized in what follows. Given a plan as a set of flexible timelines $\mathcal{PL} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$, we call *projection* a flexible plan $\mathcal{PL}' = \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}$ derived from $\mathcal{PL}$ by setting the temporal occurrence of each uncontrollable timepoint to a fixed value. In other words, a projection describes one possible uncontrollable evolution of the flexible plan. 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 with respect to 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 $p2$.

[3] We assume $|\mathcal{T}_{TL}| \geq 2$, $\mathcal{O}_{TL}(tp_0) = [t_O, t_O]$,$\mathcal{O}_{TL}(tp_{|\mathcal{T}_{TL}|}) = [t_H, t_H]$.
[4] The value $\mathcal{V}_{TL}(tp_i)$ is supposed to be taken in $[tp_i, tp_{i+1})$, the value $\mathcal{V}_{TL}(tp_{|\mathcal{T}_{TL}|})$ is undefined.
[5] $\mathcal{D}_{TL}(tp_{|\mathcal{T}_{TL}|})$ is undefined.

3. Timed Game Automata

Timed Automata provide a natural way to represent time delays of real-time systems. They allow to annotate state-transitions with temporal constraints using real-valued clock variables. A *clock* is a non-negative, 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 is an integer, i.e. $c \in \mathbb{Z}$, $x, y \in X$, 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 3.1. 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, Inv : $Q \rightarrow B(X)$ is a function associating to each location $q \in Q$ a constraint 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 $\mathbb{R}_{\geq 0}$ of nonnegative reals. We denote with $\mathbb{R}^X_{\geq 0}$ the set of valuations on X and with $\vec{0}$ the valuation that assigns the value 0 to each clock. $Y \subseteq X$ is a subset of X, i.e., a subset of clocks, we denote with $v[Y]$ the valuation which assigns the value 0 to any $z \in Y$ and agrees with $v(z)$ for any $z \in (X - Y)$. For any $\delta \in \mathbb{R}_{\geq 0}$ we denote with $(v + \delta)$ the valuation such that, for each $x \in X$, $(v + \delta)(x) = v(x) + \delta$. Let $g \in C(X)$ and v be a valuation. We say that v satisfies g, notation $v \models g$ if the constraint g evaluated on v is true. A *state* of a TA $\mathcal{A}$ is a pair (q, v) such 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) such that $v \models \text{Inv}(q)$. A *discrete transition* for $\mathcal{A}$ is a 5-tuple (q, v, a, q', v') such that $(q, v) \overset{a}{\rightarrow} (q', v')$ where $(q, v), (q', v') \in S$, $a \in$ Act, and there exists a transition $q \overset{g,a,Y}{\rightarrow} q' \in E$ such that $v \models g$, $v' = v[Y]$, $v \models \text{Inv}(q)$ and $v' \models \text{Inv}(q')$. Notice that Y represents the subset of clocks to be reset (i.e. $v' = v[Y]$) after the transition. A *time transition* for $\mathcal{A}$ is a 4-tuple (q, v, δ, v') such that $(q, v) \overset{\delta}{\rightarrow} (q, v')$ where $(q, v) \in S$, $(q, v') \in S$, $\delta \in \mathbb{R}_{\geq 0}$, $v' = v + \delta$, $v \models \text{Inv}(q)$, 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 Runs($\mathcal{A}, (q, v)$) the set of runs of $\mathcal{A}$ starting from state (q, v); we write Runs($\mathcal{A}$) for Runs($\mathcal{A}, (q_0, \vec{0})$). If ρ is a finite run we denote with last(ρ) the last state of run ρ. A ***network* of TA (nTA)** is a finite set of TA evolving in parallel with a CCS style semantics as a model for concurrency [21]. 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, but 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 = $Q_1 \times \ldots \times Q_n$. 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 $\text{Inv}(q_1, \ldots q_n) = \text{Inv}_1(q_1) \wedge \ldots \wedge \text{Inv}_n(q_n)$. 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 3.2. A **Timed Game Automaton (TGA)** [20] 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 TAs, run, configuration are easily extended to TGAs. Given a TGA $\mathcal{A}$ and three sets of locations *Init*, *Safe*, and *Goal*, the *reachability control problem*, or *reachability game*, $RG(\mathcal{A}, Init, Safe, Goal)$ 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 *memoryless* whenever its result depends only on the last configuration of the run.

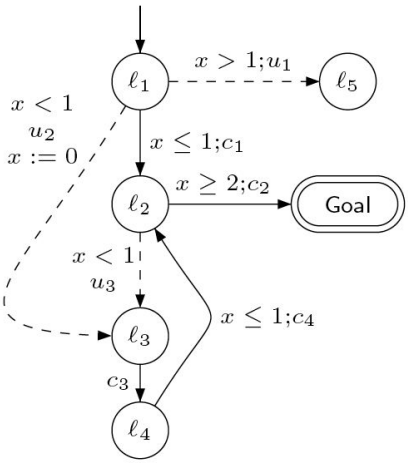

Figure 2: A timed game automaton.

As an example, consider the timed game automaton in Figure 2, taken from [3], consisting of one clock x and two types of edges: controllable (c_i) and uncontrollable (u_i). The reachability game consists in finding a strategy for a controller, i.e., when to take the controllable transitions that will guarantee that the system, regardless of when and if the opponent chooses to take uncontrollable transitions, will eventually end up in the Goal location. Obviously, for all the initial states of the form (l_1, x) with $x \leq 1$ there is such a winning strategy. For instance, a winning strategy would consist in taking c_1 immediately in all the states (l_1, x) with $x \leq 1$; taking c_2 immediately in all the states (l_2, x) with $x \geq 2$; taking c_3 immediately in all the states (l_3, x) and delaying it in all the states (l_4, x) with $x < 1$ until the value of x is 1 at which point the edge c_4 is taken.

3.1. UPPAAL-TIGA

UPPAAL-TIGA [3] extends UPPAAL [19] providing a toolbox for the specification, simulation, and verification of real-time games. It implements on-the-fly algorithms for solving games based on TGA with respect to reachability and safety properties [4]. If there is no winning strategy, UPPAAL-TIGA gives a counter strategy for the opponent (environment) to make the controller lose.

To model concurrent systems, timed automata can be extended with parallel composition. In process algebras, various parallel composition operators have been proposed to model different aspects of concurrency (e.g., CCS and CSP [21, 14]). These algebraic operators can be adopted in both TAs and TGAs.

In the UPPAAL-TIGA modeling language [19], the CCS parallel composition operator is used, which allows the interleaving of actions as well as hand-shake synchronization. Synchronous communication between the processes is based on hand-shake synchronization using input and output actions.

Given a nTGA $\mathcal{N}_A$, a set of goal states (*win*), and/or a set of bad states (*lose*), both defined by UPPAAL state formulas, four types of winning conditions can be issued [3]. For all of them, the solution of the game is a controllable strategy f such that $\mathcal{N}_A$ supervised by f ensures that: $A\Diamond\, win$ (must reach win); $A[\, not(lose)\, U\, win]$ (must reach win and must avoid lose); $A[\, not(lose)\, W\, win]$ (should reach win and must avoid lose); $A\Box\, not(lose)$ (must avoid lose). [6]

4. Building TGA from Timeline-based Planning Specifications

Our goal is to verify flexible timeline-based plans 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 modeled using nTGA. The strategy is the following: first, the state variables in the model are mapped into TGAs; second, the flexible plan is transformed in a similar way; finally, an *Observer* TGA is introduced in order to check for value constraint violations as well as synchronization violations.

4.1. Modeling a State Variable as an nTGA

Let $\mathcal{PD} = \{\mathcal{SV}_1, \ldots \mathcal{SV}_n\}$ be the set of state variables defining our planning domain. We will model each $\mathcal{SV}_i \in \mathcal{PD}$ with a TGA $\mathcal{A}_{\mathcal{SV}_i} = (Q_{\mathcal{SV}_i}, q_0, \text{Act}_{\mathcal{SV}_i}, X_{\mathcal{SV}_i}, \text{Inv}_{\mathcal{SV}_i}, E_{\mathcal{SV}_i})$. As a consequence, the set *StateVar* = $\{\mathcal{A}_{\mathcal{SV}_1}, ..., \mathcal{A}_{\mathcal{SV}_n}\}$ will represent our planning domain $\mathcal{PD}$ as a nTGA.

The TGA $\mathcal{A}_{\mathcal{SV}_i}$ is defined as follows. The set $Q_{\mathcal{SV}_i}$ of locations is the set $\mathcal{V}_i$ of values of $\mathcal{SV}_i$. The initial state q_0 is the initial value of the timeline in the plan associated to $\mathcal{SV}_i$. The set of clocks $X_{\mathcal{SV}_i}$ consists of just one local clock c_{sv_i}. The set $\text{Act}_{\mathcal{SV}_i}$ of actions corresponds to the values $\mathcal{V}_i$ of $\mathcal{SV}_i$. If $\mathcal{SV}_i$ is a planned state variable then the actions in $\text{Act}_{\mathcal{SV}_i}$ are controllable (i.e., $\text{Act}_{\mathcal{SV}_i}$ = $\text{Act}_{c\mathcal{SV}_i}$), otherwise they are uncontrollable (i.e., $\text{Act}_{\mathcal{SV}_i}$ = $\text{Act}_{u\mathcal{SV}_i}$). Location invariants $\text{Inv}_{\mathcal{SV}_i}$ are defined as follows: $\text{Inv}_{\mathcal{SV}_i}(v) := c_{sv_i} \leq u_b$, where: $v \in Q_{\mathcal{SV}_i} = \mathcal{V}_i$ and $\mathcal{D}_i(v) = [l_b, u_b]$. The transitions set $E_{\mathcal{SV}_i}$ consists of transitions of the form $v \overset{g,v'?,Y}{\rightarrow} v'$, where: $g = c_{sv_i} \geq l_b$, $Y = \{c_{sv_i}\}$, $v \in Q_{\mathcal{SV}_i} = \mathcal{V}_i$, $\mathcal{D}_i(v) = [l_b, u_b]$, $v' \in \mathcal{T}_i(v)$.

Additionally, we assume that $Act_{StateVar} = Act_{\mathcal{SV}_1} \cup ... \cup Act_{\mathcal{SV}_n}$ and $X_{StateVar} = X_{\mathcal{SV}_1} \cup ... \cup X_{\mathcal{SV}_n}$ (with $Act_{\mathcal{SV}_1} \cap ... \cap Act_{\mathcal{SV}_n} = \emptyset$ and $X_{\mathcal{SV}_1} \cap ... \cap X_{\mathcal{SV}_n} = \emptyset$).

4.2. Modeling a Flexible Plan as an nTGA

Let $\mathcal{P} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ be a flexible plan on a planning domain $\mathcal{PD}$. We will model each $\mathcal{TL}_i \in \mathcal{P}$ with a TGA $\mathcal{A}_{\mathcal{TL}_i} = (Q_{\mathcal{TL}_i}, q_0, \text{Act}_{\mathcal{TL}_i}, X_{\mathcal{TL}_i}, \text{Inv}_{\mathcal{TL}_i}, E_{\mathcal{TL}_i})$. As a consequence, the set $Plan = \{\mathcal{A}_{\mathcal{TL}_1}, ..., \mathcal{A}_{\mathcal{TL}_n}\}$ represents $\mathcal{P}$ as a nTGA.

The TGA $\mathcal{A}_{\mathcal{TL}_i}$ is defined as follows. The set $Q_{\mathcal{TL}_i}$ of locations of $\mathcal{A}_{\mathcal{TL}_i}$ consists of the set of timepoints $\mathcal{T}_{TL_i}$ in $\mathcal{TL}_i$ along with a location l_{goal} modeling the fact that the plan has been completed. Thus, $Q_{\mathcal{TL}_i} = \mathcal{T}_{TL_i} \cup \{l_{goal}\}$. The initial state q_0, of $\mathcal{A}_{\mathcal{TL}_i}$ is the first value interval l_0 in $\mathcal{TL}_i$. The set

[6] In CTL (see [12]), A means 'along All paths' (Inevitably), E means 'along at least (there Exists) one path', $\Box$ means 'has to hold on the entire subsequent path' (Globally), $\Diamond$ means 'eventually has to hold somewhere on the subsequent path' (Finally).

of clocks $X_{\mathcal{TL}_i}$ of $\mathcal{A}_{\mathcal{TL}_i}$ consists of just one element: the *plan clock* c_p. Let $\mathcal{SV}$ be the state variable that corresponds to the timeline $\mathcal{TL}$ under consideration. The set $\text{Act}_{\mathcal{TL}}$ of actions of $\mathcal{A}_{\mathcal{TL}}$ consists of the values of $\mathcal{SV}$. If $\mathcal{SV}$ is a planned state variable then the actions in $\text{Act}_{\mathcal{TL}}$ are controllable (i.e., $\text{Act}_{\mathcal{TL}}$ = $\text{Act}_{c\mathcal{TL}}$), otherwise, they are uncontrollable (i.e., $\text{Act}_{\mathcal{TL}}$ = $\text{Act}_{u\mathcal{TL}}$). Location invariants $\text{Inv}_{\mathcal{TL}}$ for $\mathcal{A}_{\mathcal{TL}}$ are defined as follows. For each $l \in \mathcal{TL}$, we consider $\mathcal{D}_{TL}(l) = [lb, ub]$ and we define $\text{Inv}_{\mathcal{TL}}(l) := c_p \leq ub$. For the goal location l_{goal} the invariant $\text{Inv}_{\mathcal{TL}}(l_{goal})$ is satisfied. The set $E_{\mathcal{TL}}$ of transitions of $\mathcal{A}_{\mathcal{TL}}$ consists of *intermediate* and *final* transitions. An intermediate transition has the form $l \overset{g,v!,Y}{\rightarrow} l'$, where: $g = c_p \geq lb$, $Y = \emptyset$ with l and l' consecutive time intervals in $\mathcal{TL}$. A final transition has the form $q \overset{\emptyset,\emptyset,\emptyset}{\rightarrow} q'$, where: $q = l_{pl}$ (pl is the plan length), $q' = l_{goal}$. Transition labels allow us to implement the hand-shake synchronization [21] between each state variable and its corresponding timeline in the plan. In fact, a synchronization between $\mathcal{A}_{\mathcal{TL}_i}$ and the related $\mathcal{A}_{\mathcal{SV}_i}$ is implemented by labeling two transitions with the same value ($v!$ and $u?$ with $u = v$). In particular, each transition of the timeline automaton $\mathcal{A}_{\mathcal{TL}_i}$ synchronizes (when this is possible) with the related transition of the state variable automaton $\mathcal{A}_{\mathcal{SV}_i}$ by means of the same label/action value.

We assume that $Act_{Plan} = Act_{\mathcal{TL}_1} \cup ... \cup Act_{\mathcal{TL}_n}$ (with $Act_{\mathcal{TL}_1} \cap ... \cap Act_{\mathcal{TL}_n} = \emptyset$), while X_{Plan} is only composed of the shared plan clock c_p.

4.3. Modeling Synchronizations with an Observer TGA

We complete the modeling of synchronizations between state variables and the $Plan$ by introducing an *Observer*, that is, a TGA reporting an error when an illegal transition occurs. Such an observer plays a relevant role in our verification procedure being deputed to error states detection.

The observer TGA $\mathcal{A}_{Obs}$ = (Q_{Obs}, q_0, Act_{Obs}, X_{Obs}, Inv_{Obs}, E_{Obs}) is defined as follows. The set of locations is $Q_{Obs} = \{l_{ok}, l_{err}\}$ modeling *legal* (l_{ok}) and *illegal* (l_{err}) executions. The initial location q_0 is l_{ok}. The set of actions is Act_{Obs} = $\{a_{fail}\}$. The set of clocks is $X_{Obs} = \{c_p\}$. There are no invariants, that is, $\text{Inv}_{Obs}(l)$ always returns the empty constraint. This models the fact that $\mathcal{A}_{Obs}$ can stay in any location indefinitely. The set E_{Obs} consists of two kinds of uncontrollable transitions: *value transitions* and *sync transitions*. Let $p \in \mathcal{TL}$ be a transition point and $\mathcal{V}_{TL}(p) = v_p$ be the associated value. A value transition has the form $l_{ok} \overset{g,a_{fail},\emptyset}{\rightarrow} l_{err}$, where: $g = \mathcal{TL}_{s_p} \wedge \neg SV_{v_p}$. Let $\langle \mathcal{SV}, v \rangle \longrightarrow \langle \{\mathcal{SV}'_1, \ldots, \mathcal{SV}'_n\}, \{v'_1, \ldots, v'_n\}, \mathcal{R} \rangle$ be a synchronization. A sync transition has the form $l_{ok} \overset{g,a_{fail},\emptyset}{\rightarrow} l_{err}$, where: $g = \neg\mathcal{R}(\mathcal{SV}_v, \mathcal{SV}'_{1\,v'_1}, \ldots, \mathcal{SV}'_{nv'_n})$. Note that for each possible cause of error (illegal value occurrence or synchronization violation) we define a suitable transition that forces our Observer TGA to move to the error location which, once reached, cannot be abandoned.

To sum up, the nTGA $\mathcal{PL}$ is composed of the set of automata $\mathcal{PL} = StateVar \cup Plan \cup \{\mathcal{A}_{Obs}\}$ which models, respectively, the State Variables, the Flexible plan under verification as well as the Domain Theory description.

Remark 4.1. It is worth noting from the description given in Section 2 that a planning domain $\mathcal{PD}$ with N state variables, each one with (at most) V allowed values and T transitions, has size $N(V+T)$. This is also the size of the nTGA *StateVar* describing $\mathcal{PD}$, defined as discussed in Section 4.1. Thus, our nTGA encoding is linear in the size of the planning domain. The size of a flexible plan $\mathcal{P}$ on $\mathcal{PD}$ (again, defined in Section 2), with N timelines and (maximal) plan length L, is NL. Using the encoding described in Section 4.2, NL is also the size of the nTGA *Plan*, describing the given flexible plan $\mathcal{P}$. Thus, our

encoding for a flexible plan is linear in the size of the flexible plan. Furthermore, the domain theory description has size S, with S the number of synchronizations. The *Observer* TGA definition given in Section 4.3 is linear in the same size S. Finally, a plan verification problem is defined considering the description of a flexible plan along with the related planning domain and domain theory. Such a description has size $N(V + T) + (NL) + S$, which is also the size of our nTGA $\mathcal{PL}$ describing the same plan verification problem. Thus, our nTGA encoding is also linear in the size of the given plan verification problem.

5. Verifying Time Flexible Plans

Given the flexible plan $\mathcal{PL}$ defined as a nTGA defined (Section 4.3), we can define a Reachability Game that guarantees, if successfully solved, the plan validity with respect to the domain constraints and the dynamic controllability property.

5.1. Plan Verification in UPPAAL-TIGA

In [8], we give a constructive proof to demonstrate that the nTGA $\mathcal{PL}$ is well defined with respect to the flexible plan $\mathcal{P}$, that is, the set of automata $\mathcal{PL}$ captures all and only the possible temporal evolutions enabled by the flexible plan $\mathcal{P}$. Hence, once we have represented a flexible plan as a nTGA, the plan verification problem can be reduced to a Reachability Game. To do this, we introduce a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of initial locations, one for each automaton in $\mathcal{PL}$, while *Safe* = $\{l_{ok}\}$ and *Goal* represents the set of goal locations, one for each $\mathcal{TL}_i$ in $\mathcal{PL}$.

To solve the reachability game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*), we can use a model checker, e.g., UPPAAL-TIGA, to check the formula $\Phi =$ *A [Safe U Goal]* in $\mathcal{PL}$. This formula requires that $\mathcal{PL}$ remains in *Safe* states until *Goal* states are reached along all the possible paths. In other words, to win the Reachability Game, the solver has to find a strategy such that, for each possible evolution of contingent transition points, all the timelines are completed and errors are avoided. Thus, if the solver can verify the above property, then the flexible temporal plan is valid (see [8] for a formal proof).

5.2. Dynamic Controllability

Finally, it is possible to show (see [8]) that if there exists a winning strategy for the Reachability Game RG, then the plan is also dynamically controllable. Indeed, recalling the *dynamic controllability* definition for timeline-base plans given in Section 2, we have that: (a) each possible evolution of the uncontrollable automata corresponds to a timeline projection p, (b) each winning strategy/solution provided by UPPAAL-TIGA for the RG corresponds to a consistent schedule T, and (c) a set of winning strategies represents a *viable execution strategy* S. Thus, any winning strategy produced by UPPAAL-TIGA represents a viable execution strategy S for the flexible plan $\mathcal{P}$. Furthermore, 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 $p2$, which implies that the flexible plan is dynamically controllable.

Remark 5.1. Note that solving a model checking problem on timed automata is a rather complex task (e.g., see [18, 30]). As a result, the computational effectiveness of our linear time translation cannot stem from a theoretical result about TGA model checking. However, as well known to the formal verification

community, many verification problem instances are actually tractable in practice. Along the same line of thinking, our experimental results in Section 7 shows that TGA verification problems, stemming from complex flexible plan verification problems, can be solved within a reasonable amount of time.

6. Case Study: the Remote Space Agent Domain

In this section, we present a specific case study that we use in our experimental analysis to generate benchmark sets and demonstrate the performances of our verifier. The domain is inspired by the MARS EXPRESS Long Term Planning Problem as described in [9, 5]. In particular, we have started from real instances of the original problem and created variants to define a generic testbed case called the "Remote Space Agent Domain".

6.1. Domain Requirements

In order to define a planning problem we consider a remote agent (e.g., a spacecraft and its control software) that operates around a target planet (e.g., Mars). The remote agent implements an internal behavior that allows it to either point to the planet and use its instruments to produce data (generically referred to as Science) or point towards a communication station (either a Relay Satellite or directly Earth) and communicate previously produced data.

The remote space agent is controlled by both a planner and an executive system to accomplish the required tasks (scientific observations, communication, and maintenance activities). These tasks have to be temporally synchronized with both the physics of the spacecraft and the exogenous events that occur independently from the agent control.

For each orbit followed by the remote space agent around the target planet, the operations are divided according to three orbital phases: (1) the pericentre (the orbital segment closest to the target planet); (2) the apocentre (the orbital segment farthest from the planet); (3) the orbital segments between the pericentre and apocentre. Around pericentre, the agent should point towards the planet, thus allowing observations of the planet surface generically referred to as Science operations. Between pericentre and apocentre, the agent should point to Earth for transmitting data. In order to modify the pointing direction (from Earth to the target planet and viceversa), the remote space agent must execute a slew manoeuvre. Communication with Earth should occur within a ground-station availability time window. Ground-station visibility can either partially overlap with or fully contain a pericentre passage. Maintenance operations should occur around the vicinity of apocentre.

The remote agent is also endowed with a set of scientific instruments or payloads (e.g., stereo cameras, altimeters, spectrometers, etc.) whose activities are to be planned during the pericentre phase taking into account physical constraints. In particular, here we are assuming that instruments can be activated one at a time through the following fixed operation sequence: warm-up, process, turn-off. Additionally, there are a number of *hard* and *soft* constraints that should be satisfied. Constraints on uplink windows frequency and duration require four hours uplink time for each 24 hours (hard constraint), and these uplink windows must be as regular as possible, i.e., one about every 20 hours (this is formulated as a soft constraint since uplink windows require ground station availability, and it is generally impossible to state exactly their positions). Moreover, it should also be possible to divide a four-hour uplink window in two two-hour uplink windows. Apocentre slots for spacecraft maintenance windows must be allocated

between 2 and 5 orbits apart (hard constraint), and the maintenance duration must be of 90 minutes (to be centered around the apocentre event).

6.2. Timeline-based Specification

To obtain a timeline-based specification of the domain we use the two types of state variables: *Planned State Variables* that represent the timelines controlled by the planning agent, and *External State Variables* that represent imposed values which can only be observed.

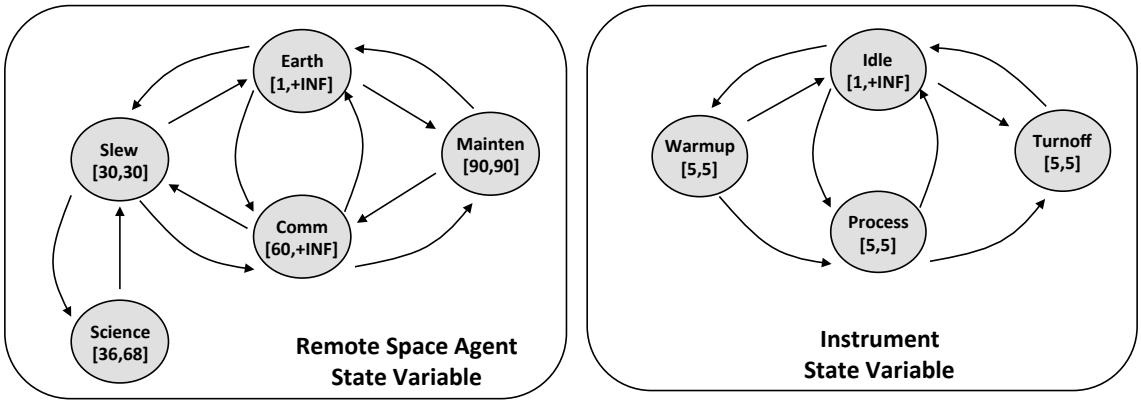

Figure 3: Value transitions for the *planned state variables* describing the Spacecraft Operative Mode (left) and any of the Instruments (right) correct behavior (Temporal durations are stated in minutes).

Planned State Variables. Planned state variables represent time varying features related to science and maintenance operations as well as the agent's communication activities. In particular, the state variable *Spacecraft Operative Mode* specifies the observation, communication, and maintenance opportunities for the agent. In Figure 3-left, we detail the values that can be taken by this state variable, their durations, and the allowed value transitions in accordance with the mission requirements and the spacecraft physics. Here, both scientific observations (*Science*) and communications (*Comm*) require a *Slew* operation to turn the remote agent towards, respectively, the target planet and the earth; the mainenance (*Mainten*) operations are allowed while the remote agent is pointing towards the earth, hence after *Earth* or after *Comm*.

Additional planned state variables, called *Instrument-1*..., *Instrument-n*, are introduced to represent the scientific payloads. In the case study, we assume that each instrument can either be idle or executing one of the tasks (warm-up, process, turn-off). Therefore, for each variable *Instrument-i* we introduce four values: *Warmup*, *Process*, *Turnoff*, and *Idle*. Figure 3-right illustrates the allowed transitions: the instrument *Warmup* is enabled from *Idle* and followed by *Process* which meets a *Turnoff* that is followed by *Idle*.

External State Variables. We use two state variables to model exogenous constraints for the planning problem. They represent contingent events, such as orbit events, or communication opportunity windows. The *Orbit Events* state variable (Figure 4, top) maintains the temporal occurrences of pericentres

and apocentres represented by the values: *Peri* and *Apo* (with fixed durations). The *Ground Station Availability* state variables (Figure 4, bottom) are a family of variables that model the visibility of several ground stations. The allowed values for these state variables are either *Available* or *Unavailable*.

Synchronizations constraints. We use Figure 4 to exemplify the use of synchronizations in our benchmark domain. Any valid temporal plan requires synchronizations between the planned timelines (see Figure 4, middle) and the external timelines (represented as dotted arrows in Figure 4). Such synchronizations specify how (a) science operations must occur during pericentres, i.e., the *Science* value must start and end within a *Peri* value; (b) maintenance operations must occur in coincidence with the apocentres time intervals, i.e., the *Maint* value is required to start and end exactly when the *Apo* value starts and ends; (c) communications must occur during ground station visibility windows, i.e., the *Comm* value must start and end during an *Available* value on any of the ground stations. As for the scientific instruments, we introduce the following constraints: (d) if *Instrument-i* is not *Idle* then the other instruments need to be *Idle*; (e) the *Warmup* value occurs before *Process* which is before *Turnoff*; (f) the latter activities are allowed only when *Science* is active along the *Operative Mode* timeline.

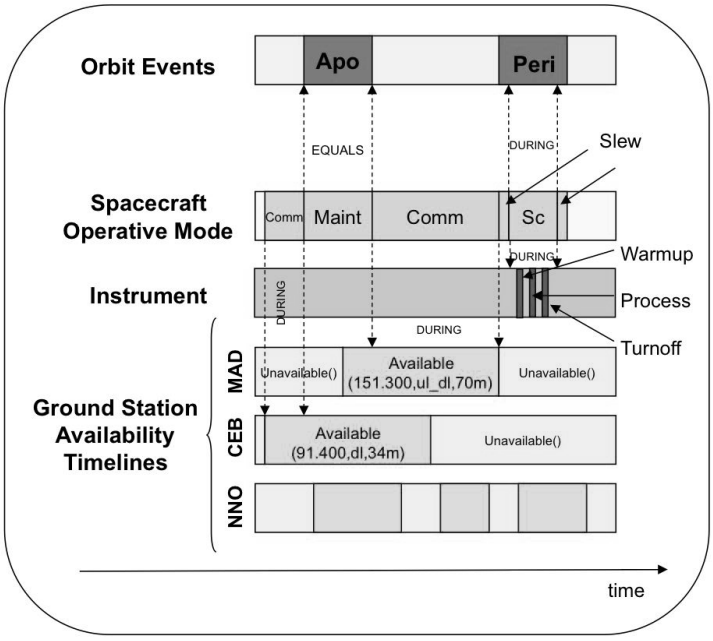

Figure 4: An example of complete plan for the Remote Space Agent domain. The synchronizations among timelines are highlighted.

In addition to those synchronization constraints, the *Operative Mode* timeline must respect transition constraints among values and durations for each value specified in the domain (see again Figure 3-left).

Relaxed constraints. It is worth reminding that in addressing real applications other constraints that cannot be naturally represented as temporal synchronizations among specific activities might have to be

considered besides ordinary synchronization constraints. Nevertheless, these constraints, that we call *relaxed constraints*, define a kind of desiderata on the global behavior of the generated plan. These requirements are not explicitly represented in the planning model as structural constraints, but they are rather treated as meta-level requirements to be enforced by the planner heuristics and optimization methods. In our case study, we consider the following relaxed constraints: (g) *Maint* must be allocated between 2 and 5 orbits apart with duration of about 90 minutes (to be centered around the apocentre event); (h) the amount of science activities must be maximized, i.e., a *Science* event should occur during each pericentre phase.

6.3. Temporal Planning

Given the set of requirements described above, a planning activity for the remote agent domain must produce the following decisions on the timelines: selection of the Maintenance windows; selection of the Communication windows among the set of available ground stations visibility windows; selection of the windows for Science operations, around pericentre events; selection of the operation activities for the scientific instruments.

In this domain, we exploit a *least-commitment* temporal planner that produces a flexible temporal plan along the given planning horizon, generally covering hundreds of orbits. In particular, in our framework the generated plan is obtained by means of a hybrid approach that combines two solving methods: a general purpose timeline planning & scheduling system generates a partial solution that is then refined by a domain dependent solver to provide the final plan (the approach is similar to the hybrid solver described in [5]). The partial plan generated by the first solver has to satisfy all the synchronization constraints explicitly represented in the planning domain. The second solver exploits some heuristics and optimization methods to complete the partial plan, and attempts to enforce the remaining relaxed constraints.

Once a solution is obtained, we are interested in verifying two aspects of the solution:

1. **Robustness and Controllability.** Our planning system generates temporally flexible plans by means of a least-commitment temporal planning approach that does not discriminate between controllable and uncontrollable activities; hence, the generated plan cannot directly ensure *dynamic controllability* (see Section 2). On the other hand, a temporally flexible plan provides robustness and adaptability with respect to uncontrollable events because it can be adapted on-line by the executive system. Therefore, an important issue is to check whether plan *robustness* provided by time flexibility can also entail *dynamic controllability*.

2. **Relaxed vs. Synchronization Constraints.** As explained above, in our domain specification we distinguish between *synchronization constraints* and *relaxed constraints*. While the first ones are explicitly represented in the model as structural constraints, the second ones are treated as implicit requirements to be addressed by heuristics and optimization methods. This means that these *relaxed constraints* cannot be guaranteed by the planning process, hence plan verification is required to check whether they are satisfied.

6.4. Domain Specification in UPPAAL-TIGA

Given the Remote Space Agent Domain we now show how it can be automatically represented in the specification language of UPPAAL-TIGA following the encoding introduced in Section 4.

```
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 5: Module definition for the Spacecraft Operative Module.

For each state variable, we introduce a *state variable timed automaton* whose locations correspond to the values of the state variable, while transitions represent changes of values. For example, in Figure 5 we have the encoding for the Spacecraft Operative Mode state variable (here called `process REMOTE_AGT()`) where the states are: `Earth`, `Earth_Comm`, `Science`, `Maintenance`, and `Slew`. Temporal constraints are specified by value duration constraints (in terms of $[min, max]$) and sequencing constraints between values (expressed by temporal relations). The duration constraints (e.g., Science activity duration in $[36, 68]$, see Figure 3) are encoded as clock invariants and guards on the outgoing transitions. For example, for Science we have `Science clockREMOTE_AGT <= 4080` and `guard clockREMOTE_AGT >= 2160` (values stated in seconds). Sequencing constraints, e.g., Science precedes Slew, are encoded as outgoing transitions, e.g., `Science -> Slew` in Figure 5.

The input model for the verifier includes also the encoding of the flexible plan. A temporal plan describes the sequence of values the state variables should assume within the planning horizon. To represent flexible plans, we introduce a general *plan clock* and an automaton for each planned timeline. For example, in Figure 7 we find a graphical representation of two TIGA automata representing two planned timelines: a) represents the planned activities for the Spacecraft Operative Module; b) represents the evolution of the ground station availability state variable. Each timeline automaton has a number of

states that equates the length of the plan: one state for each activation/decision available in the plan, plus a final special state *goal*, which represents the plan completion.

```
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));  },
    ...
    OK -u-> ERR { guard ((INSTRUMENTWarmup)
                        and not (REMOTE_AGTScience));  },
    OK -u-> ERR { guard ((INSTRUMENTProcess)
                        and not (REMOTE_AGTScience));  },
    OK -u-> ERR { guard ((INSTRUMENTTurnoff)
                        and not (REMOTE_AGTScience));  },
    ...

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

Figure 6: Partial monitor module definition. Note that Monitor is uncontrollable.

The plan steps are represented by transitions. For example in Figure 7 dotted and continuous transitions represent uncontrollable and controllable transitions respectively; the transitions along the automaton a) represent the activations of Slew (`pulse Slew!`) and Science (`pulse Science!`). For each transition, we introduce a guard that implements the minimum duration for the activity (e.g., for Slew we have `clockPlan >= 12439`) and an invariant that represents the maximal allowed duration (e.g., for Slew we have `clockPlan <= 14259`). Here, the synchronization channels are exploited to link the transitions of the planned timelines to the transitions of the state variables automata. For instance, in Figure 7, the first controllable transition labeled `pulse Slew!` shall synchronize with the transition labeled `pulse Slew?` for the related REMOTE_AGT automaton (see Figure 5). For instance, the second transition in Figure 7a synchronizes with the related transition defined in Figure 5 between Slew and Science locations.

Finally, we define a fully uncontrollable *observer automaton* (See Fig. 6). Such a monitor checks the consistency of temporal constraints, that is, it checks value sequencing and synchronizations constraints.

Thus, given the above specified input specification, we can 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 external timelines, the goal must be reached while avoiding monitor errors. If this is verified, UPPAAL-TIGA returns a control execution strategy, such that, when correctly executed, guarantees to fulfill the planning goal. That is, verifying the above property implies validating the flexible temporal plan. Furthermore, since the input model encompasses all the temporal constraints of the domain, the UPPAAL-TIGA verification algorithm guarantees that all

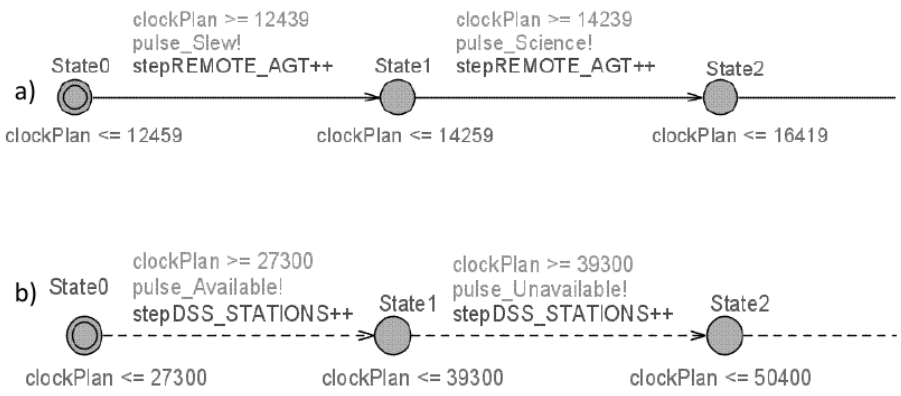

Figure 7: TIGA models for the timelines: a) planned (controllable) activities for the Spacecraft Operative Module state variable; b) evolution (uncontrollable) of the ground station availability state variable.

the time points in the strategy only depend on the past events. This property is needed to ensure the *dynamic controllability* of the flexible temporal plan (see also [8]).

7. Empirical Results

The main goal of this work is not to assess the worst case complexity of our approach, but rather to show its practical feasibility by evaluating the effectiveness of our method when deployed on challenging real world problems. In fact, model checking is PSPACE complete in the size of the system state space [26] while the number of system states is exponential in the size of the program that defines the system (our input). As a result, even a polynomial verification algorithm (the best we can hope for, by [26]) will take an exponential amount of memory. Indeed, model checking success rests on the availability of verification algorithms that can successfully exploit the specific structure of the verification problem at hand in order to counteract (i.e., delay) *state explosion*. Accordingly, for us the issue is not *if* we will run out-of-memory (we know that this will always happen), but rather *when* we will run out-of-memory. In fact, this will give us an insight about the *feasibility* of the proposed approach in our context. More specifically, it will show the size of the systems we can hope to handle for the class of verification problems we are considering. For this reason we will usually carry out verification experiments with systems of increasing size (along some system parameter) until we run out-of-memory (or hit some given memory or time threshold). Thus, in order to evaluate feasibility of our approach, in this section we present and discuss experimental results about verification of the flexible plans generated for our case study. In particular, we analyze the plan verification performances with respect to temporal *flexibility* and execution *controllability*.

7.1. Evaluation Strategy

In order to empirically demonstrate the feasibility of our verification method, here we deploy it in different configurations of the benchmark domain. In this way we propose various executive contexts

for checking both *dynamic controllability* and *relaxed constraints* satisfaction. We analyze the performances of our tool in scenarios obtained by varying: the number of state variables/timelines considered; plan length; temporal flexibility. Moreover, we consider incrementally complex execution contexts obtained by reducing the controllable activities while increasing the uncontrollable ones. In particular, we consider the following settings:

– *State variables.* The remote agent can be endowed with different scientific instruments. Here, we consider configurations with zero, one, two, three or more scientific instruments. This affects the number of state variables, activities, and synchronization constraints.

– *Flexibility.* For each scientific instrument activity (i.e., warm-up, process, turnoff), we set a minimal duration (i.e., about 2 minutes), but we allow temporal flexibility on the activity termination, namely, the end of each activity has a tolerance ranging from 5 to 10 seconds. E.g., if we set 5 seconds of flexibility, we introduce an uncertainty on the activity terminations, for instance, the warmup activity can take from 120 to 125 seconds. This temporal interval represents the degree of temporal flexibility that we introduce in the system.

– *Horizon.* We consider flexible plans with a horizon length ranging from 3 to 10 mission days.

– *Controllability.* We consider four different executive contexts: (1) all the instruments activities are controllable; (2) for each instrument the warmup termination is not controllable; (3) for each instrument, warmup and process terminations are not controllable; (4) for each instrument warmup, process, and turnoff are not controllable.

Note that the higher the degree of flexibility/uncontrollability, the larger is the space of allowed behaviors to be checked, thus, the more demandability plan verification process.

Within these experimental settings, we analyze the performance of our tool considering the issues of: model generation; dynamic controllability checking; domain requirements checking.

– *Model generation.* A preliminary analysis concerns the model generation process and the dimension of the generated UPPAAL-TIGA specification. This analysis is necessary because the complexity of UPPAAL-TIGA model generation affects the scalability of the overall verification method.

– *Dynamic controllability checking.* We analyze the plan verification performances while checking the dynamic controllability. In particular, we collect the time performances (CPU time) of plan verification in different scenarios (changing the degree of plan flexibility) and executive contexts (changing the plan controllability).

– *Relaxed constraints checking.* We check for plan correctness with respect to the *relaxed domain* constraints (see Section 6) that we consider of interest.

We run our experiments on a Linux workstation endowed with a 64-bit AMD Athlon CPU (3.5GHz) and 2GB RAM. In the following we illustrate the collected empirical results (timings are reported in seconds).

7.2. Model Generation

The first step of our verification process consist of the UPPAAL-TIGA model generation. For this purpose, we have developed a tool that automatically builds the UPPAAL-TIGA model given the description of the *planning domain* and the *flexible temporal plan* to be checked. Here, we want to assess the size of the generated model and the generation time with respect to the dimension of the planning domain and of the plan (state variables and plan length). In our experimental setting, we consider domain models with an incremental number of state variables (from 3 to 6) and plans with an incremental number of mission days (from 3 to 10). For each possible configuration, we consider the size of the generated model and the time elapsed for the generation. For all these configurations, we observe that the generation process is very fast, taking less than $250ms$, while the dimension of the generated model gradually grows with respect to the dimension of the flexible plan (in terms of number of timelines and plan length).

	3 timelines		4 timelines		5 timelines		6 timelines	
days	kb	nr. states	kb	nr. states	kb	nr. states	kb	nr. states
3	16	41	19	51	23	61	59	136
4	32	85	38	110	42	135	62	90
5	54	131	58	179	63	227	86	242
6	73	168	77	240	82	312	100	297
7	94	204	98	300	101	396	129	497
8	107	238	112	351	117	464	148	600
9	119	271	125	397	130	523	169	727
10	139	301	142	439	147	577	191	855

Figure 8: Size of the generated model (kb and number of states) with respect to the plan length and number of timelines.

In conclusion, the process of model generation is shown to be fast as the generated model grows linearly with the dimension of the plan. As a consequence the encoding phase is not a critical step. This provides an empirical support for Remark 4.1.

7.3. Flexible Plan Verification against Fully Controllable Execution

In this section, we analyze the impact of temporal flexibility on the verification process in the easiest condition for controllability. Indeed, in this initial experimental setting, we consider fully controllable plans assuming all the scientific tasks to be controllable. In this context, we check for dynamic controllability and assess the plan verification performances as the plan flexibility is gradually increased.

Figures 9(a), 10(a) and 11(a) illustrate the results obtained in the case of one, two and three instruments, respectively. In the three tables, we consider the verifier performances under different plan length and flexibility conditions. Here, $0s$ flex means that all the temporal variables in the plan are completely instantiated, while $5s$ (or $10s$) flex means that the non-instantiated temporal variables in the plan spans an interval of $5s$ (or $10s$) seconds.

The results show that an increment of temporal flexibility has a limited impact on the performances of the verification tool. This is particularly evident in the case of a single instrument, where the performances of the verification process seems not to be affected by the degree of temporal flexibility (Figure 9(a)). On the other hand, as the number of scientific instruments increases (Figure 10(a) and

Full Controllability			
days	0s flex	5s flex	10s flex
3	0,198	0,202	0,254
4	0,254	0,301	0,320
5	0,300	0,344	0,328
6	0,192	0,208	0,184
7	0,248	0,240	0,248
8	0,292	0,300	0,284
9	0,348	0,332	0,364
10	0,392	0,364	0,401

(a)

1 Uncontrollable Task			
days	0s flex	5s flex	10s flex
3	0,189	0,165	0,193
4	0,227	0,234	0,238
5	0,276	0,296	0,264
6	0,172	0,160	0,168
7	0,212	0,220	0,208
8	0,268	0,248	0,252
9	0,308	0,336	0,336
10	0,356	0,364	0,379

(b)

2 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	0,189	0,192	0,188
4	0,246	0,237	0,245
5	0,296	0,324	0,288
6	0,156	0,164	0,164
7	0,212	0,216	0,212
8	0,260	0,263	0,264
9	0,316	0,288	0,336
10	0,345	0,321	0,335

(c)

3 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	0,198	0,221	0,212
4	0,267	0,283	0,267
5	0,304	0,288	0,288
6	0,188	0,172	0,176
7	0,212	0,208	0,220
8	0,252	0,236	0,248
9	0,312	0,300	0,332
10	0,367	0,353	0,379

(d)

Figure 9: Verification with one instrument varying flexibility and controllability.

Full Controllability			
days	0s flex	5s flex	10s flex
3	0,899	2,010	2,673
4	1,123	3,101	3,200
5	1,664	3,508	3,312
6	2,756	3,780	3,396
7	3,704	4,368	4,528
8	4,492	5,080	5,088
9	5,300	5,896	6,724
10	5,934	6,234	7,243

(a)

1 Uncontrollable Task			
days	0s flex	5s flex	10s flex
3	1,784	2,998	3,021
4	2,132	3,156	3,103
5	2,784	3,280	3,248
6	2,892	3,252	3,312
7	3,664	4,384	4,500
8	4,232	5,096	5,212
9	5,492	6,492	6,716
10	6,357	7,093	7,732

(b)

2 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	2,022	3,105	3,227
4	2,214	3,326	3,339
5	2,444	3,452	3,548
6	2,652	3,212	3,328
7	3,612	4,412	4,464
8	4,200	4,879	5,208
9	5,300	5,876	6,812
10	6,604	7,012	8,002

(c)

3 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	2,243	3,143	3,004
4	2,527	3,340	3,122
5	2,880	3,528	3,052
6	2,628	3,404	3,704
7	3,604	4,252	4,284
8	4,212	4,668	4,98
9	5,176	6,088	6,384
10	6,392	7,478	8,244

(d)

Figure 10: Verification with two instruments changing both flexibility and controllability.

Figure 11(a)), we can observe a smooth growth of the verification time with respect to the allowed temporal flexibility. Of course, this is mainly due to the fact that in this case the verification process has to check all the synchronization constraints among the instruments, which are not considered in the case of a single instrument. However, even though the increment of temporal flexibility widens the spectrum of possible behaviors to be checked, in the presence of fully controllable activities a single execution trace is sufficient to show plan controllability; hence the verification task is reduced to checking for correct plan termination.

Full Controllability			
days	0s flex	5s flex	10s flex
3	30,68	73,93	75,03
4	84,63	201,34	198,85
5	158,85	424,52	422,72
6	238,12	583,27	587,84
7	258,36	644,19	625,91
8	270,28	661,79	635,90
9	353,33	874,16	817,43
10	404,72	1000,64	925,16

(a)

1 Uncontrollable Task			
days	0s flex	5s flex	10s flex
3	30,95	72,55	71,33
4	82,53	211,78	190,19
5	156,21	445,31	418,74
6	238,29	567,50	578,89
7	264,42	622,41	606,65
8	275,22	646,70	617,43
9	359,51	850,19	827,87
10	412,34	970,90	946,16

(b)

2 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	30,37	70,67	70,79
4	81,83	212,14	199,23
5	154,08	416,47	416,82
6	234,26	558,08	574,96
7	254,76	620,09	603,00
8	266,23	643,57	613,80
9	348,08	843,17	821,55
10	398,89	964,02	938,25

(c)

3 Uncontrollable Tasks			
days	0s flex	5s flex	10s flex
3	30,69	69,87	70,04
4	80,89	210,77	188,25
5	154,77	414,96	415,84
6	235,95	556,23	573,54
7	255,52	617,76	600,95
8	266,66	640,05	610,50
9	349,24	839,59	819,66
10	400,24	959,96	936,75

(d)

Figure 11: Verification with three instruments varying both flexibility and controllability.

7.4. Flexible Plan Verification against Partially Controllable Execution

We consider now the verifier performances while checking the dynamic controllability in the presence of uncontrollable activities. In the case of a single scientific instrument, the obtained results (see Figures 9b-c-d) are comparable with the ones collected in the fully controllable case. Even when we consider a setting where all the tasks are uncontrollable, our verification tool can easily accomplish plan verification for all the flexibility and plan length configurations (see Figure 9(d)). In the case of 2 instruments (hence, 5 timelines), the raise of temporal flexibility gradually increases the time required by the verification tool to verify the plans (see Figures 10b-c-d). A similar increment can be observed when we increase the number of uncontrollable activities. If we keep the number of uncontrollable activities constant, the performance trend appears to be similar to the one of the fully controllable case. An analogous trend can be observed in Figure 11(b-c-d) when we consider 3 instruments (6 timelines). If we consider the worst case, i.e., all the activities uncontrollable and maximal temporal flexibility with 2 instruments, the per-

formances of the UPPAAL-TIGA verification tool are still satisfactory: given flexible plans with horizon length up to 10 mission days and 5 timelines, plan verification can be successfully accomplished within a few seconds (see Figure 10(d)). On the other hand, when we consider 3 instruments, the performances are significantly reduced (see Figure 11(d)). Moreover, with 4 additional instruments (i.e., 7 timelines) the problem becomes very hard. In particular, we found that UPPAAL-TIGA is not able to verify settings with four and five scientific instruments throwing out-of-memory exceptions. Solutions can be obtained for less than three mission days, but at the cost of several hours of computation time. Therefore, the number of scientific instruments appears to be a critical dimension in our benchmark.

The increase of computational time due to each additional instruments is illustrated in both Figure 12 and Figure 13 where we can observe a clear discontinuity in time processing for each new timeline. In our case, such discontinuity is emphasized by the high tasks density associated with each timeline. This can be observed in Figure 14 where we show, for each scenario, the number of temporal flexible tasks that have to be checked (most of which are to be checked in all the possible temporal permutations) in order to validate the associated flexible plans.

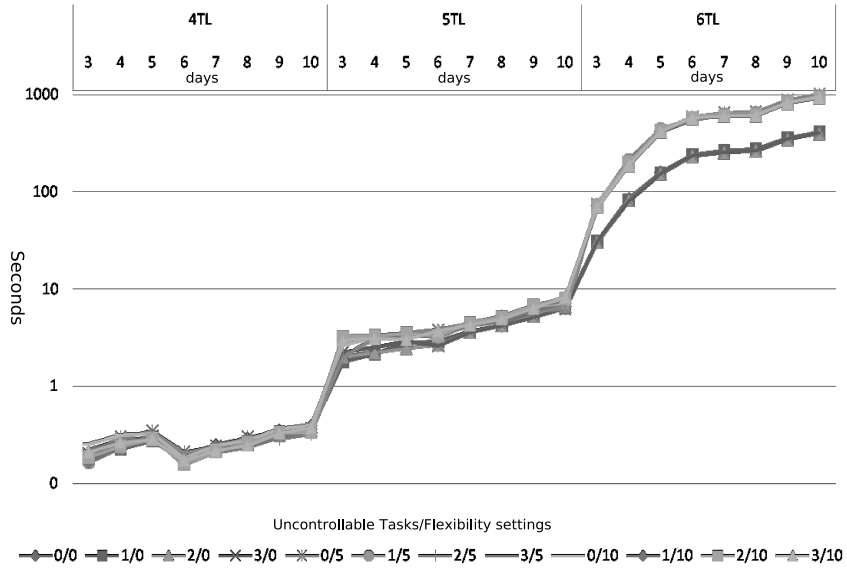

Figure 12: Computation time with respect to horizon length and number of timelines.

On the other hand, when we consider the computational time with respect to the increase in flexibility and controllability (Figure 13), we can see that the performance is pretty stable with a little change due to the increase of flexibility in the plans. The following results not only demonstrate the feasibility of our verification method in a real world scenario, but also provide us with an upper bound for the performance of our tool in this benchmark domain.

7.5. Flexible Plan Verification against Relaxed Domain Constraints

We also performed some experiments to verify not only the dynamic controllability, but also other domain-dependent constraints, namely, the two *relaxed constraints* introduced in Section 6.2:

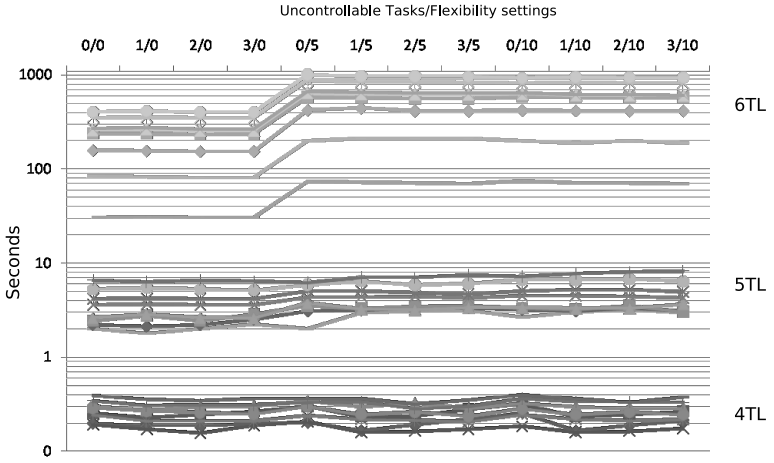

Figure 13: Computation time with respect to controllability and flexibility.

	TL3	TL4	TL5	TL6	TL7
3	51	69	87	105	123
4	67	98	130	161	193
5	100	149	199	248	298
6	121	175	229	283	337
7	150	222	294	366	438
8	175	256	337	418	499
9	200	295	390	485	580
10	225	333	441	548	656

Figure 14: Number of tasks depending on the timelines and the horizon.

– every 2 and 5 orbits a maintenance activity should be performed;

– during each pericentre, a science activity should be performed.

In this experimental setting we assume that the system is equipped with 2 scientific instruments (5 timelines). In Figure 15, we report the experimental results obtained by increasing the degree of uncontrollability on the considered flexible plans.

When we change the plan flexibility, the verifier exhibits performances that are analogous to those reported in the previous cases. That is, the additional properties to be checked carry a low additional overhead to the verification process.

As a final remark, the experimental results presented here demonstrate the feasibility and the effectiveness of our method. Notwithstanding the limited number of timelines processed, our case study scenarios are pretty complex. Indeed, they are obtained from a real world application scenario whose complexity is augmented with additional (and fictitious) timelines. In this context, we found that the number of instruments has significant impact on the verifier performances, which is emphasized by the density of the timelines (Figure 14). On the other hand, the parallel increase of temporal flexibility and plan uncontrollability does not entail the presumably expected computational overhead.

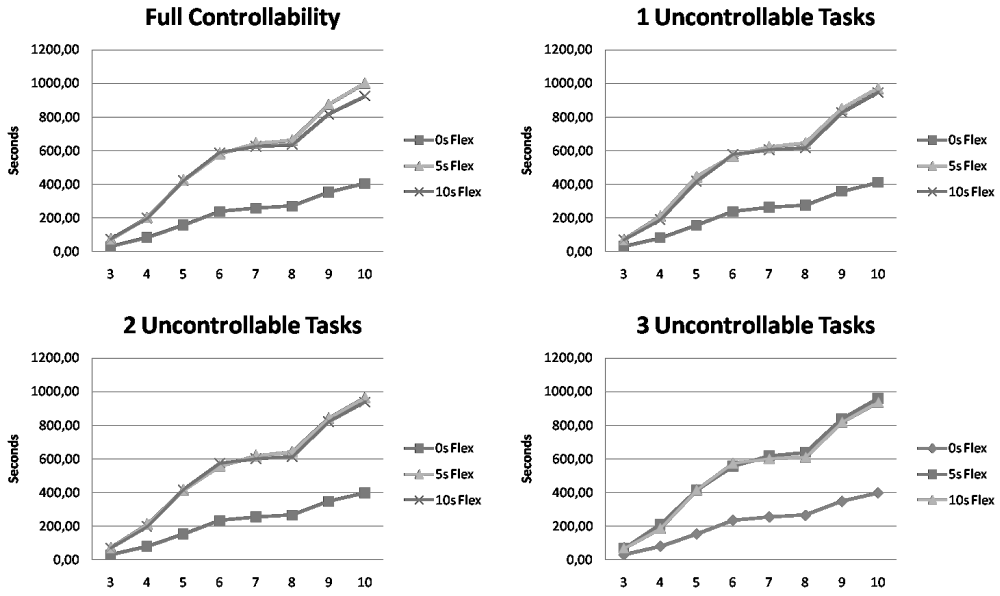

Figure 15: Plan verification against dynamic controllability and relaxed constraints: two instruments.

8. Conclusions

This work addresses the problem of flexible temporal plan verification deploying model-checking on timed game automata. As well known, because of state explosion [26] a model checking problem on timed automata is generally very complex (e.g., see [18, 30]). Fortunately, not all instances of verification problems fall in the worst case category. Indeed, UPPAAL-TIGA yields very encouraging performance results on some interesting classes of verification problems [4].

The paper presents and discusses some experimental results stemming from interesting flexible plan verification problems. It introduceds a benchmark domain obtained from a real-world space application whose complexity has been augmented by means of additional features. In this context, we have assessed the performances of the approach along different coordinates: number of timelines, plan length, number of uncontrollable activities, temporal flexibility. The collected results show that interesting real-world instances of dynamic controllability checking and domain dependent verification problems can be effectively solved using UPPAAL-TIGA. Our empirical analysis is proposed as a first benchmark for flexible temporal plan verification.

Acknowledgements

Authors would like to thank the reviewers for their comments and criticisms that pushed us to extend the results and improve the presentation. Authors also thank Riccardo Rasconi for carefully proofreading the final version of the paper.

References

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

[2] Alur, R., Dill, D. L.: A Theory of Timed Automata, *Theoretical Computer Science*, **126**, 1994, 183–235.

[3] Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K. G., Lime, D.: UPPAAL-TIGA: Time for Playing Games!, in: *CAV-07. Proc. of Computer Aided Verification*, number 4590 in Lecture Notes in Computer Science, Springer, 2007, 121–125.

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

[5] Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: MRSPOCK: Steps in Developing an End-to-End Space Application, *Computational Intelligence*, 2010, Accepted for publication.

[6] Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Flexible Timeline-Based Plan Verification, in: *KI-09*, vol. 5803 of *Lecture Notes in Artificial Intelligence*, Springer, 2009, 49–56.

[7] Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Validation and Verification Issues in a Timeline-Based Planning System, *Knowledge Engineering Review*, 2010, 25, pp 299-318.

[8] Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Analyzing Flexible Timeline-Based Plans, *ECAI-10. Procedings of the 19th European Conference on Artificial Intelligence, Lisbon, Portugal*, August 2010.

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

[10] Cesta, A., Oddi, A.: DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains,, in: *New Directions in AI Planning* (M. Ghallab, A. Milani, Eds.), IOS Press: Amsterdam, 1996, 341–452.

[11] Chien, S., Tran, D., Rabideau, G., Schaffer, S., Mandl, D., Frye, S.: Timeline-Based Space Operations Scheduling with External Constraints, *ICAPS-10. Proc. of the Twentieth International Conference on Automated Planning and Scheduling*, 2010.

[12] Clarke, E. M., Grumberg, O., Peled, D. A.: *Model Checking*, The MIT Press, 1999.

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

[14] Hoare, C. A. R.: Communicating Sequential Processes, *Commun. ACM*, **26**(1), 1983, 100–106.

[15] Howey, R., Long, D.: VAL's Progress: The Automatic Validation Tool for PDDL2.1 Used in the International Planning Competition, *Proc. of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*, Trento, Italy, June 2003.

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

[17] Khatib, L., Muscettola, N., Havelund, K.: Mapping Temporal Planning Constraints into Timed Automata, *TIME-01. The Eigth Int. Symposium on Temporal Representation and Reasoning*, 2001.

[18] Larsen, K., Pettersson, P., Yi, W.: Compositional and Symbolic Model-Checking of Real-time Systems, *Real-Time Systems Symposium, IEEE International*, **0**, 1995, 76, ISSN 1052-8725.

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

[20] Maler, O., Pnueli, A., Sifakis, J.: On the Synthesis of Discrete Controllers for Timed Systems, *STACS*, LNCS, Springer, 1995.

[21] Milner, R.: *Communication and Concurrency*, Prentice-Hall, Inc., 1989, ISBN 0-13-115007-3.

[22] Morris, P. H., Muscettola, N.: Temporal Dynamic Controllability Revisited, *AAAI-05. Proc. of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, Pittsburgh, Pennsylvania, USA*, 2005.

[23] Morris, P. H., Muscettola, N., Vidal, T.: Dynamic Control of Plans With Temporal Uncertainty, *IJCAI-01. Proc. of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA*, 2001.

[24] Muscettola, N.: HSTS: Integrating Planning and Scheduling, in: *Intelligent Scheduling* (Zweben, M. and Fox, M.S., Ed.), Morgan Kauffmann, 1994, 169–212.

[25] Muscettola, N., Smith, S., Cesta, A., D'Aloisi, D.: Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture, *IEEE Control Systems*, **12**(1), 1992, 28–37.

[26] Sistla, A. P., Clarke, E. M.: The Complexity of Propositional Linear Temporal Logics, *J. ACM*, **32**(3), 1985, 733–749, ISSN 0004-5411.

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

[28] Vidal, T.: Controllability Characterization and Checking in Contingent Temporal Constraint Networks, *KR–00. Principles of Knowledge Representation and Reasoning Proc. of the Seventh International Conference, Breckenridge, Colorado, USA*, 2000.

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

[30] Yovine, S.: Model Checking Timed Automata, in: *Lectures on Embedded Systems*, vol. 1494 of *Lecture Notes in Computer Science*, Springer, 1998, 114–152.