How Hard is Verifying Flexible Temporal Plans for the Remote Space Agent?

A. Cesta[†] and **A. Finzi**[‡] and **S. Fratini**[†] and **A. Orlandini**[†] and **E. Tronci**[§]
[†] ISTC-CNR, Via S.Martino della Battaglia 44, I-00185 Rome, Italy
[‡] DSF “Federico II” University, Via Cinthia, I-80126 Naples, Italy
[§] DI “La Sapienza” University, Via Salaria 198, I-00198 Rome, Italy

Abstract

Timeline-based planners have been shown quite successful in addressing real world problems. Nevertheless they represent a niche technology in AI P&S research as an application synthesis with such techniques is still considered a sort of “black art”. Our current work aims at both creating a rational reference architecture for timeline-based planning and scheduling and developing a knowledge engineering environment around such problem solving tool. In particular we are integrating verification tools in such engineering environment to enhance typical capabilities of a constraint-based planner. In this paper we present recent results on the connection between plan generation and execution from a particular perspective: the static verification of plans before their execution. In particular, we present a verification process suitable for a timeline-based planner and show how a temporally flexible plan verification problem can be cast as model-checking on timed game automata. We here discuss the effectiveness of the proposed approach in a thorough experimental analysis based on a realistic domain called “The Remote Space Agent”.

Introduction

In the past, several planning systems were endowed with development environments to facilitate application design (e.g., O-PLAN (Tate, Drabble, and Kirby 1994)). More recent examples of software development environments are EUROPA (EUROPA 2008) and ASPEN (Sherwood et al. 2000). Such environments can be enriched in several directions. In a recent work (Cesta et al. 2010b), these authors have envisaged the synthesis of knowledge engineering environments in which constraint-based and validation and verification techniques concur in creating an enhanced software environment for P&S. In particular, we are working on verification and validation methods for timeline-based planning investigating the use of model checking techniques for verifying properties of specific planning software applications.

An important problem in timeline-based planning as used in (Muscettola 1994; EUROPA 2008; Sherwood et al. 2000) is the connection with plan execution which is instrumental in several challenging real domain (e.g., the aspect is relevant for both autonomy in space and robotics). Broadly speaking such architectures return an envelope of potential solutions in form of a flexible plan which is commonly accepted to be less brittle of a single plan when coping with execution. But the general formal properties of such a representation are far from being statically defined. Some aspects of such plans have been studied by working on the temporal network which is underlying the constraint based plan representation often used by such systems – see for example (Vidal and Fargier 1999; Morris and Muscettola 2005). We have addressed the more general question of verifying flexible plans working on the more abstract plan view as set of timelines with formal tools like model checkers.

These authors have been investigating one aspect which we consider as missing: the interconnection between timeline-based planning and standard techniques for formal validation and verification (V&V). The broad aim here is the one of building a powerful environment for knowledge engineering (Cesta et al. 2010b) and also that of exploring properties that concern temporal plans and their execution (Cesta et al. 2009a; 2009b). In particular, (Cesta et al. 2009a) provides a feasibility study for the approach, while, in (Cesta et al. 2009b), some formal properties are further investigated. In this paper we mainly address a limitation of (Cesta et al. 2009b): the fact that experiments were very preliminary.

Here, that work is carried on by: (a) introducing a benchmark problem which is realistic and rich enough to allow experiments along different directions; (b) presenting a complete experimental analysis considering incrementally complex scenarios and configurations in the benchmark domain. The collected results show that the approach based on model checking can be effective in practice. Indeed, despite the increasing complexity of the verification tests, the verifier performances remain acceptable for static analysis in a knowledge engineering environment.

Preliminaries

This section shortly present the two basic ingredients we combine in our knowledge engineering environment: timeline-based planning and timed game automata. It is worth mentioning that in (Abdedaim et al. 2007) the same ingredients are put together for a different purpose than ours, namely the mapping from temporal constraint-based planning problems into UPPAAL-TIGA game-reachability problems.

Timeline-Based Planning and Execution

Timeline-based planning is an approach to temporal planning which has been applied in the solution of several real world problems – e.g., (Muscettola 1994). The approach pursues a general idea that planning and scheduling consist in the synthesis of desired temporal behavior for complex physical systems. In this respect, the set of features of a domain that needs control are modeled as a set of temporal functions whose values over a time horizon have to be planned for. 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, the time varying features are called multi-valued *state variables* as in (Muscettola 1994). As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal evolution of timelines. Such causal laws are specified for the state variables in a *domain specification* which specifies the operational constraints in 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 desired set of evolutions always satisfying the domain specification.

We assume that the temporal features we want to represent as state-variables have a finite set of possible values assumed over temporal intervals. The temporal evolutions 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 $[lb, ub]$). (b) and (c) specify the operational constraints on the values in (a).

In this type of planning, a *planning domain* is defined as a set of state variables $\{\mathcal{SV}_1, \ldots, \mathcal{SV}_n\}$. They cannot be considered as reciprocally decoupled but a set of additional relations exist, called *synchronizations*, modeling 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

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

where: $\mathcal{TL}$ is the reference timeline; v is a value on $\mathcal{TL}$ which makes the synchronization applicable; $\{ \langle \mathcal{TL}'_1, v'_1 \rangle \ldots, \langle \mathcal{TL}'_n, v'_n \rangle \}$ 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.

Timeline based planning. The temporal evolutions of a state variable will be described by means of *timelines*, that is a sequence of state variable values, a set of ordered transition points between the values and a set of distance constraints between transition points. When the transition points are bounded by the planning process (lower and upper bounds are given for them) instead of being exactly specified, as it happens in case of a least commitment solving approach for instance, we refer to the timeline as *time flexible* and to the plan resulting from a set of flexible timeline as a *flexible plan*.

It is worth mentioning that *planning goals* are expressed as desiderata of values in temporal intervals and the task of the planner is to build timelines that describe valid sequences of values that achieve the desiderata.

A *plan* is defined as a set of timelines $\{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ over the same interval for each state variable. The process of *solution extraction* from a plan is the process of computing (if exists) a *valid* and completely specified set of timelines from a given set of time-flexible timelines. A solution is *valid* with respect to a domain theory if every temporal occurrence of a reference value implies that the related target values hold on target timelines presenting temporal intervals that satisfy the expected relations.

Plan execution. During plan execution the plan is under responsibility to an executive program that forces value transitions over timeline. A well known problem with execution is that not all the value transitions are under responsibility of the executive but event exists that are under control of *nature*. As a consequence, an executive cannot completely predict the behavior of the controlled physical system because the duration of certain processes or the timing of exogenous events is outside of its control. In such cases, the values for the state variables that are under the executive scope should be chosen so that they do not constrain uncontrollable events. This is the *controllability problem* defined, for example, in (Vidal and Fargier 1999) where *contingent* and *executable* processes are distinguished. The contingent processes are not controllable, hence with uncertain durations, instead the executable processes are started and ended by the executive system. Controllability issues underlying a plan representation have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) representation in (Vidal and Fargier 1999) where basic formal notions are given for *dynamic* controllability (see also (Morris and Muscettola 2005)). In 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 $p2$.

Timed Game Automata

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

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

A *state* of a TGA is a pair $(q, v) \in Q \times R^X_{\geq 0}$ that consists of a discrete part and a valuation of the clocks (i.e., a value assignment for each clock in X). An *admissible* state for a $\mathcal{A}$ is a state (q, v) s.t. $v \models \mathrm{Inv}(q)$. From a state (q, v) a TGA can either let time progress or do a discrete transition and reach a new state.

A *time transition* for $\mathcal{A}$ is 4-tuple $(q, v) \overset{\delta}{\rightarrow} (q, v')$ where $(q, v) \in S$, $(q, v') \in S$, $\delta \in R_{\geq 0}$, $v' = v + \delta$, $v \models \mathrm{Inv}(q)$ and $v' \models \mathrm{Inv}(q)$. That is, in a time transition a TGA does not change location, but only its clock values. Note that all clock variables are incremented by the same amount δ in valuation v'. This is why variables in X are named *clocks*. Accordingly, δ models the *elapsed time* during the time transition.

A *discrete transition* for $\mathcal{A}$ is 5-tuple $(q, v) \overset{a}{\rightarrow} (q', v')$ where $(q, v) \in S$, $(q', v') \in S$, $a \in$ Act and there exists a transition $q \overset{g,a,Y}{\rightarrow} q' \in E$ s.t. $v \models g$, $v' = v[Y]$ and $v' \models \mathrm{Inv}(q')$. In other words, there is a discrete transition (labeled with a) from state (q, v) to state (q', v') if the clock values (valuation v) satisfy the *transition guard* g and the clock values after resetting the clocks in Y (valuation v') satisfy the invariant of location q'. Note that an admissible transition always leads to an admissible state and that only clocks in Y (reset clocks) change their value (namely, to 0).

A *run* of a TGA $\mathcal{A}$ is a finite or infinite sequence of alternating time and discrete transitions of $\mathcal{A}$. We denote with $\mathrm{Runs}(\mathcal{A}, (q, v))$ the set of runs of $\mathcal{A}$ starting from state (q, v) and write $\mathrm{Runs}(\mathcal{A})$ for $\mathrm{Runs}(\mathcal{A}, (q, \vec{0}))$. If ρ is a finite run, we denote with $\mathrm{last}(\rho)$ the last state of run ρ and with $\mathrm{Duration}(\rho)$ the sum of the elapsed times of all time transitions in ρ.

A *network* of TGA (nTGA) is a finite set of TGA evolving in parallel with a CSS style semantics for parallelism. Namely, at any time, only one TGA in the network can change location, unless a synchronization on labels takes place. In the latter case, the two automata synchronizing on the same label move together. Note that time does not elapse during synchronizations.

Given a TGA $\mathcal{A}$ and three symbolic configurations *Init*, *Safe*, and *Goal*, the *reachability control problem* or reachability game $RG(\mathcal{A}$, *Init*, *Safe*, *Goal*$)$ consists in finding a *strategy* f such that $\mathcal{A}$ starting from *Init* and supervised by f generates a winning run that stays in *Safe* and enforces *Goal*.

A strategy is a partial mapping f from the set of runs of $\mathcal{A}$ starting from *Init* to the set $\mathrm{Act}_c \cup \{\lambda\}$ (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 $f(\rho) = l$.

Using nTGA to model timeline-based planning specifications

Timed Game Automata are particularly suitable for modeling controllability problems because the uncontrollable activities can be modeled as adversary moves. Following this approach, we perform flexible timeline-based plan verification 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. Our strategy is the following. First, timelines and state variables are mapped to TGA. Second, we model the flexible plan *view* of the world by partitioning state variables/timelines into two classes: controllable and uncontrollable. Finally, an *Observer* TGA is introduced in order to check for value constraints violations as well as synchronizations violations.

Modeling a Planning Domain 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 $\mathcal{SV} \in \mathcal{PD}$ with a TGA $\mathcal{A}_{SV}$ = (Q_{SV}, q_0, Act_{SV}, X_{SV}, Inv_{SV}, E_{SV}). Then the set $SV = \{\mathcal{A}_{SV_1}, ..., \mathcal{A}_{SV_n}\}$ represents our planning domain $\mathcal{PD}$ as an nTGA.

The TGA $\mathcal{A}_{SV}$ is defined as follows. The set Q_{SV} of locations of $\mathcal{A}_{SV}$ is just the set $\mathcal{V}$ of values of $\mathcal{SV}$. The initial state q_0, of $\mathcal{A}_{SV}$ is the initial value in the timeline of $\mathcal{SV}$. The set of clocks X_{SV} of $\mathcal{A}_{SV}$ consists of just one local clock: c_{sv}. The set Act_{SV} of actions of $\mathcal{A}_{SV}$ consists of the values $\mathcal{V}$ of $\mathcal{SV}$. If $\mathcal{SV}$ is controllable then the actions in Act_{SV} are controllable (i.e., Act_{SV} = Act_{cSV}), otherwise they are uncontrollable (i.e., Act_{SV} = Act_{uSV}). Location invariants Inv_{SV} for $\mathcal{A}_{SV}$ are defined as follows: $\mathrm{Inv}_{SV}(v) := c_{sv} \leq u_b$, where: $v \in Q_{SV} = \mathcal{V}$ and $\mathcal{D}(v)$ = $[l_b, u_b]$. The set E_{SV} of transitions of $\mathcal{A}_{SV}$ consists of transitions of the form $v \overset{g,v'?,Y}{\rightarrow} v'$, where: $g = c_{sv} \geq l_b$, $Y = \{c_{sv}\}$, $v \in Q_{SV} = \mathcal{V}$, $\mathcal{D}(v)$ = $[l_b, u_b]$, $v' \in \mathcal{T}(v)$.

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

The TGA $\mathcal{A}_{\mathcal{TL}}$ is defined as follows. The set $Q_{\mathcal{TL}}$ of locations of $\mathcal{A}_{\mathcal{TL}}$ consists of the value intervals (*plan steps*) in $\mathcal{TL}$ along with a location l_{goal} modeling the fact that the plan has been completed. Thus, $Q_{\mathcal{TL}} = \mathcal{TL} \cup \{l_{goal}\}$. The initial state q_0, of $\mathcal{A}_{\mathcal{TL}}$ is the first value interval l_0 in $\mathcal{TL}$. The set of clocks $X_{\mathcal{TL}}$ of $\mathcal{A}_{\mathcal{TL}}$ consists of just one element: the *plan clock* c_p. Let $\mathcal{SV}$ be the state variable corresponding 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 controllable 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 $l = [lb, ub] \in \mathcal{TL}$ 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 identically true, modeling the fact that once plan is completed we can stay there as long as we like. The set $E_{\mathcal{TL}}$ of transitions of $\mathcal{A}_{\mathcal{TL}}$ consists of *intermediate* and *final* transitions. An intermediate transitions 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), $q' = l_{goal}$. Note how using state variable values as transitions label we implement the synchronization between state variables and planned timelines.

Modeling Synchronizations with an Observer TGA. We model synchronization between SV and $Plan$ with an *Observer*, that is a TGA reporting an error when an illegal transition occurs.

The observer TGA $\mathcal{A}_{Obs} = (Q_{Obs}, q_0, \text{Act}_{Obs}, X_{Obs}, \text{Inv}_{Obs}, E_{Obs})$ 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 $\text{Act}_{Obs} = \{a_{fail}\}$. The set of clocks is $X_{Obs} = \{c_p\}$. There are no invariants, that is $\text{Inv}_{Obs}(l)$ returns always the empty constraint. This models the fact that $\mathcal{A}_{Obs}$ can stay in any location as long as it likes. The set E_{Obs} consists of two kind of uncontrollable transitions: *value transitions* and *sync transitions*. Let $s_p \in \mathcal{TL}$ be a plan step and $v_p \in \mathcal{SV}$ its associated planned 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 $\langle \mathcal{TL}, v\rangle \longrightarrow \langle \{\mathcal{TL}'_1, \dots, \mathcal{TL}'_n\}, \{v'_1, \dots, v'_n\}, \mathcal{R}\rangle$ be a synchronization. A sync transition has the form $l_{ok} \overset{g,a_{fail},\emptyset}{\rightarrow} l_{err}$, where: $g = \neg\mathcal{R}(\mathcal{TL}_v, \mathcal{TL}'_{1v'_1}, \dots, \mathcal{TL}'_{nv'_n})$. Note how, for each possible cause of error (illegal value occurrence or synchronization violation), a suitable transition is defined, forcing our Observer TGA to move to the error location which, once reached, cannot be left.

The nTGA $\mathcal{PL}$ composed by the set of automata $PL = SV \cup Plan \cup \{\mathcal{A}_{Obs}\}$ models Flexible plan, State Variables and Domain Theory descriptions.

Time flexible plan verification

Given the nTGA $\mathcal{PL}$ defined above, we can define a Reachability Game that ensures, once successfully solved, the plan validity with respect to all the domain constraints and dynamic controllability.

nTGA and Flexible Plans

In (Cesta et al. 2009b), we demonstrated by construction that we obtain a one-to-one mapping between flexible behaviors, defined by $\mathcal{P}$, and automata behaviors, defined by $\mathcal{PL}$, with the Observer automaton holding the error location if either an illegal value occurs or a synchronization is violated. More specifically, it is possible to show that the set of automata $Plan = \{\mathcal{A}_{\mathcal{TL}_1}, ..., \mathcal{A}_{\mathcal{TL}_n}\}$ captures all and only the possible evolutions enabled by the flexible plan $\mathcal{P}$, that is: each automaton $\mathcal{A}_{\mathcal{TL}_i}$ describes the sequence of values for the $\mathcal{TL}_i$ timeline within the planning horizon $\mathcal{H}$; by construction, each automata in $SV = \{\mathcal{A}_{SV_1}, ..., \mathcal{A}_{SV_n}\}$ represent the associated state variable in one-to-one correspondence; finally, the Observer automaton checks for both values consistency (between planned timelines and state variables) and synchronizations satisfaction.

Plan Verification in UPPAAL-TIGA

Once we have represented flexible plans as nTGA, the plan verification problem can be reduced to a Reachability Game.

For this purpose, we introduce a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of initial locations, one for each automaton in $\mathcal{PL}$, *Safe* = $\{l_{ok}\}$, and *Goal* is for the set of goal locations, one for each $\mathcal{TL}_i$ in $\mathcal{PL}$.

In order to solve $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*), we use UPPAAL-TIGA (Behrmann et al. 2007). This tool extends UPPAAL (Larsen, Pettersson, and Yi 1997) 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 (Behrmann et al. 2007). Then, to solve the reachability game, we ask UPPAAL-TIGA to check the formula $\Phi =$ *A [Safe U Goal]* in $\mathcal{PL}$. In fact, this formula means that along all the possible paths, $\mathcal{PL}$ remains in *Safe* states until *Goal* states are reached. Thus, if the solver can verify the above property, then the flexible temporal plan is valid (again, see (Cesta et al. 2009b) for a formal account).

Whenever the flexible plan is not verified, UPPAAL-TIGA produces an execution strategy showing one temporal evolution that leads to a fault. Such a strategy can be exploited in order to understand whether the plan has some weakness or flaws are present in the planning model. In (Cesta et al. 2010b), the authors address this issue in a more general way.

Dynamic Controllability

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 timelines introduced in the second section, we can notice that each possible evolution of the uncontrollable automata corresponds to a timeline projection p. Each strategy/solution for the RG corresponds to a consistent schedule T and a set of strategy represents a viable execution strategy S. Thus, the winning strategies produced by UPPAAL-TIGA represents a viable execution strategy S for the flexible plan $\mathcal{P}$. Furthermore, the use of forward algorithms (Behrmann et al. 2007) 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.

A new benchmark domain

An aspect worth being addressed is the following: does the method have any practical relevance? In this respect, we have investigated the possibility of tailoring our method in order to implement a realistic benchmark, collect a set of experimental results and show its actual feasibility.

In this section, we present a case study that we use in our experimental analysis. The domain is inspired by a Space Mission Long Term Planning problem as described in (Cesta et al. 2008; 2010a).

We consider a remote space agent (RSA) that operates around a target planet. The RSA can either point to the planet and use its instruments to produce scientific data or point towards a communication station (Relay Satellite or Earth) and communicate previously produced data. The RSA is controlled by a planner and an executive system to accomplish the required tasks (scientific observations, communication, and maintenance activities). For each orbit followed by the RSA around the planet, the operations are split with respect to 3 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 toward the planet, thus allowing observations of the planet surface (Science operations). Between pericentre and apocentre passages, the agent should point to Earth for transmitting data. Communication with Earth should occur within a ground-station availability window. Ground-station visibility can either partially overlap or fully contain a pericentre passage. Maintenance operations should occur around the apocentre passages. The RSA is also endowed with a set of scientific instruments or payloads (e.g., stereo cameras, altimeters, spectrometers, etc.) whose activities are to be planned for during the pericentre phase taking into account physical constraints. In particular here we are assuming that instruments can be activated one at a time by following a fixed execution sequence of operations: warm-up, process, turn-off. Additionally, there are other constraints to be satisfied. Constraints on uplink windows frequency and duration require four hours uplink time for each 24 hours, and these uplink windows must be as regular as possible, one every about 20 hours. Apocentre slots for spacecraft maintenance windows must be allocated between 2 and 5 orbits apart, and the maintenance duration is of 90 minutes.

Timeline-based Specification. To obtain a timeline-based specification of the domain we use: *Planned State Variables* representing the timelines where there are activities under the agent control (they are planned for by the agent); *External State Variables*, representing values imposed over time which can only be observed

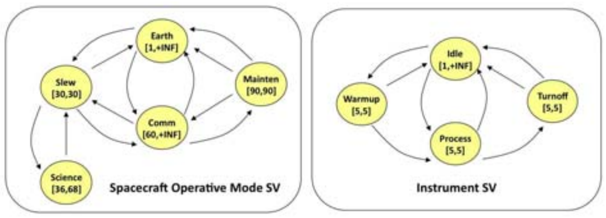

Figure 1: Value transitions for the *planned state variables* describing the Spacecraft Operative Mode (left) and any of the Instruments (right) correct behavior.

Planned State Variables. A state variable *Spacecraft Operative Mode* specifies the observation, communication, and maintenance opportunities for the agent. In Figure 1-left, we detail the values that can be taken by this state variable, their durations, and the allowed value transitions. Additional planned state variables, called *Instrument-1*..., *Instrument-n*, are introduced to represent the scientific payloads. For each variable *Instrument-i* we introduce four values: *Warmup*, *Process*, *Turnoff*, and *Idle* (see Figure 1-right).

External State Variables. The *Orbit Events* state variable (Figure 2, top) maintains the temporal occurrences of pericentres and apocentres represented by the values: *PERI* and *APO* (they have fixed durations). The *Ground Station Availability* state variables (Figure 2, bottom) are a family of variables that maintain the visibility of various ground stations. The allowed values for these state variables are either *Available* or *Unavailable*.

Synchronizations constraints. Any valid temporal plan needs synchronizations among the planned timelines (see Figure 2, middle) and the external timelines (represented as dotted arrows in Figure 2). They represent how (a) science operations must occur during pericentres, i.e., the *Science* value must start and end during a *Peri* value; (b) maintenance operations must occur in the same time interval as apocentres, 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 scientific instruments, we introduce the following constraints: (d) if *Instrument-i* is not in *Idle* then the other instruments need to be in *Idle*; (e) the *Warmup* is before *Process* which is before *Turnoff*; (f) these activities are allowed only when *Science* is active along the *Operative Mode* timeline.

Relaxed constraints. Besides synchronization constraints, we need to take into account other constraints which cannot be naturally represented in the planning model as structural constraints, but 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

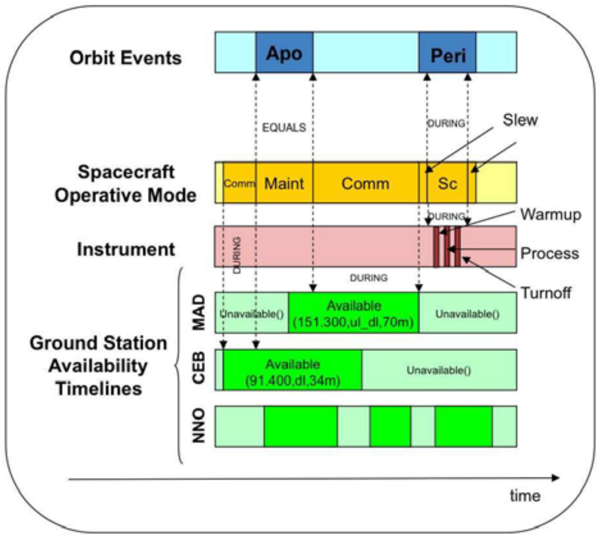

Figure 2: An example of complete plan for the Remote Space Agent domain. The synchronizations among timelines are highlighted.

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) science activities must be maximized, i.e., during each pericentre phase a *Science* event should occur.

Experimental evaluation

In this work, we demonstrate the feasibility of our approach. In particular, in this section, we analyze the plan verification performances with respect to temporal *flexibility* and execution *controllability*. In particular, we deploy our verifier in different scenarios and execution contexts checking for dynamic controllability and relaxed constraints satisfaction.

More specifically, we analyze the performances of our tool varying the following settings: *State variables.* Here, we consider three possible configurations: the RSA endowed with zero, one, or two scientific instruments. This affects the number of state variables (and synchronization constraints). *Flexibility.* For each scientific instrument activity (i.e., warm-up, process, turn-off), 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 warm-up 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 execution contexts: 1) all the instruments activities are controllable; 2) for each instrument the warm-up termination is not controllable; 3) for each instrument, warm-up and process terminations are not controllable; 4) for each instrument warm-up, process, and turn-off are not controllable.

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

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

Model Generation. A first, preliminary, analysis concerns the model generation process and the dimension of the generated UPPAAL-TIGA specification. This analysis is needed because the complexity of the generated UPPAAL-TIGA models can affect the scalability of the overall verification method. In fact, for this purpose, we developed a tool that implements the nTGA modeling procedure described before (see Section "‘Using nTga to model timeline-based planning specifications"’) and 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 5) and plans with an incremental number of mission days (from 3 to 10). For each possible configuration, we consider the dimension of the generated model and the time elapsed for the generation. For all these configurations, the generation process is very fast and takes less than $200ms$, 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	
days	kb	nr. states	kb	nr. states	kb	nr. states
3	16	41	19	51	23	61
4	32	85	38	110	42	135
5	54	131	58	179	63	227
6	73	168	77	240	82	312
7	94	204	98	300	101	396
8	107	238	112	351	117	464
9	119	271	125	397	130	523
10	139	301	142	439	147	577

Figure 3: 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 fast and the generated model grows linearly with the dimension of the plan, therefore, here the encoding phase is not a critical step.

Flexible Plan Verification against Fully Controllable Execution. Here, we collect the time performances (CPU time) of plan verification in different scenarios (changing the degree of plan flexibility) and execution contexts (changing the plan controllability).

Here, we analyze the plan verification performances in checking dynamic controllability in the easiest condition of controllability. Indeed, in this initial experimental setting, we consider fully controllable plans assuming all the scientific tasks to be controllable.

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 4: Verification with one additional instrument varying flexibility and controllability.

In Figure 4(a) and Figure 5(a), we illustrate the results gathered in the case of one and two instruments, respectively, considering the verifier performances under different plan length and flexibility conditions. The results in Figure 4(a) and Figure 5(a) 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 affected by the degree of temporal flexibility (Figures 4(a)). On the other hand, in the case of 2 scientific instruments (Figures 4(b)), 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 is to check all the synchronization constraints among the instruments, which are not considered in the case of a single instrument. However, even thought the increment of temporal flexibility enlarges the number 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 correct plan termination checking.

Flexible Plan Verification against Partially Controllable Execution. In the following, we consider the verifier performances in checking *dynamic controllability* in the presence of uncontrollable activities. Interestingly, also in this setting the execution time for verification grows in a gradual manner. In the case of a single scientific instrument, the gathered results (see Figures 4b-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 4(d)). In the case of 2 instruments (hence, 5 timelines), the increment of flexibility gradually increments the time

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 5: Verification with two instruments changing both flexibility and controllability.

needed by the verification tool to verify the plans (see Figures 5b-c-d). A similar increment can be observed when we increase the number of uncontrollable activities. If we keep constant the uncontrollable activities, the performances trend appears similar to the one of the fully controllable case. Nevertheless, even if we consider the worst case, i.e. all the activities uncontrollable and maximal temporal flexibility, the performances of the UPPAAL-TIGA verification tool are still very satisfactory: given flexible plans with horizon length up to 10 mission days and 5 timelines, plan verification can be successfully accomplished within few seconds (see Figure 5(d)).

Flexible Plan Verification against Relaxed Domain Constraints. We also perform tests to verify also other domain-dependent constraints, namely, the two *relaxed constraints* on maintenance and science activities introduced in the previous section. In this experimental setting, we assume the system endowed with 2 scientific instruments (5 timelines). In Figure 6, we report the experimental results collected increasing the degree of uncontrollability on the considered flexible plans.
Changing the plan flexibility, the verifier presents performances that are analogous to the ones reported in the previous case. Thus, the additional properties to be checked provide a low additional overhead to the verification process.

Conclusion

In our path to enhancing a knowledge engineering environment for timeline-based problem solving, we are investigating the integration of formal methods as a way of orthogonal contribution to analyze properties of plans. In recent work including the current one we are proposing the combined use of timeline-based planning and standard techniques for formal validation and verification. In particular, we have synthesized a verification process suitable for a timeline-based planner showing how a temporally flexible plan verification problem can be cast as model-checking on timed game au-

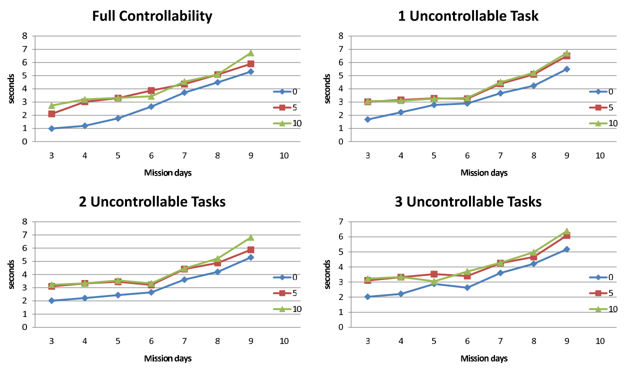

Figure 6: Experimental results collected validating flexible plans varying flexibility and controllability in case study with two additional instruments.

tomata. Then, we have investigated the possibility of tailoring our method in order to implement a realistic benchmark, collect a set of experimental results and show its actual feasibility. The experimental results presented in this paper demonstrate the feasibility of our method and the effectiveness of UPPAAL-TIGA in this setting. In fact, despite the increasing complexity of the verification configurations, the execution time gradually grows with the complexity of the task. Furthermore, the concurrent increase of temporal flexibility and plan uncontrollability does not determine the expected computational overhead. The UPPAAL-TIGA verifier can effectively handle the flexible plan verification task in all the considered configurations.

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 has been also partially supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI).

References

Abdedaim, Y.; Asarin, E.; Gallien, M.; Ingrand, F.; Lesire, C.; and Sighireanu, M. 2007. Planning Robust Temporal Plans: A Comparison Between CBTP and TGA Approaches. In *Proc. of the 7th International Conference on Automated Planning and Scheduling*, 2–10.

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

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

Cesta, A.; Fratini, S.; Oddi, A.; and Pecora, F. 2008. 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.

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

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

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

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010b. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review*. Accepted for publication.

EUROPA. 2008. Europa Software Distribution Web Site. https://babelfish.arc.nasa.gov/trac/europa/.

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

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

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

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

Sherwood, R.; Engelhardt, B.; Rabideau, G.; Chien, S.; and Knight, R. 2000. ASPEN, Automatic Scheduling and Planning Environment. Technical Report D-15482, JPL.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-Plan2: An Open Architecture for Command, Planning, and Control. In Zweben, M., and Fox, S. M., eds., *Intelligent Scheduling*. Morgan Kaufmann.

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