Verifying Flexible Timeline-Based Plans

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
* DIA “Roma TRE” University, Via della Vasca Navale 79, I-00146 Rome, Italy
§ DI “La Sapienza” University, Via Salaria 198, I-00198 Rome, Italy

Abstract

The synthesis of flexible temporal plans has demonstrated wide applications possibilities in heterogeneous domains. We are currently studying the connection between plan generation and execution from the particular perspective of verifying a flexible plan before actual execution. This paper explores how a model-checking verification tool, based on UPPAAL-TIGA, is suitable for verifying flexible temporal plans. We first describe the formal model, the formalism, and the verification method. Furthermore we discuss our own approach and some preliminary empirical results using a real-world case study.

Introduction

Timeline-based planning has been shown very effective for applications in heterogeneous real-world domains – see (Muscettola 1994; Jonsson et al. 2000; Frank and Jonsson 2003; Smith, Frank, and Jonsson 2003). A problem for a wider diffusion of such technology stems in the limited community that has been studying formal properties of this planning approach.

We are currently working at investigating the interconnection between timeline-based planning and standard techniques for formal validation and verification. In an initial work (Cesta et al. 2009b), we have listed several directions for contamination between the two technologies, then we have started addressing properties to develop a robust environment for plan generation and execution. In particular, among several V&V tasks, (Cesta et al. 2009b) identifies plan verification as a crucial task and proposes a generic model checking approach to accomplish such a task.

Here, we propose a formal account of more recent work focusing on formal verification of flexible temporal plans. Such a task can be deployed at different levels: namely, to validate either domain models or the planner, to verify the plan before execution, etc. The main contribution of the present paper is in presenting a formalization used for verification of flexible temporal plans that make use of Timed Game Automata (Maler, Pnueli, and Sifakis 1995) and UPPAAL-TIGA (Behrmann et al. 2007), a well known model-checking tool. Then, the paper describes the verification method, presenting the exploited formalism and providing current results on its usage.

It is worth noting that such an approach allows us to apply our V&V method on any timeline-based P&S system (EUROPA (Frank and Jonsson 2003), IDEA (Jonsson et al. 2000), APSI-TRF (Cesta and Fratini 2008), etc.) and even on flexible temporal plans manually generated/modified (e.g., as done on MERs (Bresina et al. 2004)). In this sense, our V&V method can be considered general while relies on an independent checker (with respect to planners’ logic/reasoning/tool).

Moreover, to show the feasibility and effectiveness of the approach we illustrate how the controllability problem (Vidal and Fargier 1999, Morris, Muscettola, and Vidal 2001) can be encoded and solved by deploying the proposed methodology. In real domains, the controllability problem arises when a generated temporally flexible plan is to be executed by an executive system that manages controllable processes in presence of exogenous events. In this scenario, the duration of the execution process is not completely under the control of the executive: the actions that are under the scope of the executive should be chosen so that they do not constrain uncontrollable events. Since (Vidal and Fargier 1999) the problem of controllability has been addressed through the temporal network which underlies a temporal plan representation, here we show how our general purpose verification method can be deployed to solve this relevant problem in flexible plan verification.

Related works. Closely related to our work is (Abdedaim et al. 2007), which proposes a mapping from temporal constraint-based planning problems into UPPAAL-TIGA game-reachability problems and presents a comparison of the two planning approaches. Authors main concern was plan synthesis, while our current goal is flexible plans verification. The approach to problem modeling is similar, however, in that work the flexibility issue remains open. Also (Khatib, Muscettola, and Havelund 2001) propose a mapping from interval-based temporal relations models (i.e., Domain Description Language models from RAX-PS) to timed automata models of UPPAAL (Larsen, Pettersson, and Yi 1997), but again flexible timeline verification was not addressed. Furthermore, (Vidal 2000) proposes a mapping from Contingent Temporal Constraint Networks (a generalization of STPUs) to Timed Game Automata which is analogous to the one exploited here. In this work, the use of
a model checker is suggested only to obtain a more compact representation and not to verify plan properties. In a PDDL framework, (Howey and Long 2003) tackle verification of temporal plans, however, authors do not address flexible temporal plans, and more expressive temporal features.

**Timeline-Based Planning and Execution**

Timeline-based planning is an approach to temporal planning (Muscettola 1994) where the generated plans are represented by sets of timelines. Each timeline denotes the evolution of a particular feature in a dynamic system. A planning domain encodes the possible evolutions of the timelines whose time points have to satisfy temporal constraints, usually represented as Simple Temporal Problem (STP) restrictions.

Here, we assume that the timelines in a planning domain are incarnations of multi-valued state variables as in (Muscettola 1994). A state variable is characterized by a finite set of values describing its temporal evolutions, and by minimal and maximal duration for each value. More formally, a state variable is defined by a tuple $(\mathcal{V}, \mathcal{T}, \mathcal{D})$ where:

- $(\mathcal{V} = \{v_1, \ldots, v_n\})$ is a finite set of values;
- $(\mathcal{T} : \mathcal{V} \to 2^\mathcal{T})$ is the value transition function;
- $(\mathcal{D} : \mathcal{V} \to \mathbb{N} \times \mathbb{N})$ is the value duration function.

Given a state variable, its associated timeline is represented as a sequence of values in the temporal interval $\mathcal{H} = [0, H]$. Each value satisfies previous (a-b-c) specifications and is defined on a set of non-overlapping time intervals contained in $\mathcal{H}$. We suppose that adjacent intervals present different values. A timeline is said completely specified over the temporal horizon $\mathcal{H}$ if there is a sequence of non-overlapping valid intervals exists and its union is equal to $\mathcal{H}$. A timeline is said time-flexible when it is completely specified and transition events are associated to temporal intervals (lower and upper bounds are given for them), instead of exact temporal occurrences. In other words, a time-flexible timeline represents a set of timelines, all sharing the same sequence of values. It is worth noting that not all the timelines in this set are valid (satisfies a-b-c). The process of timeline extraction from a time-flexible timeline is the process of computing (if exists) a valid and completely specified timeline from a given time-flexible timeline. In timeline-based planning, a planning domain is defined as a set of state variables $\{S\mathcal{V}_1, \ldots, S\mathcal{V}_n\}$ that cannot be considered as reciprocally decoupled. Then, a domain theory is defined as a set of additional relations, called synchronizations, that model the existing temporal constraints among state variables. A synchronization has the form $(\mathcal{T}_{\mathcal{L}}, v) \leadsto (\{\mathcal{T}_{\mathcal{L}_1}, \ldots, \mathcal{T}_{\mathcal{L}_n}\}, \{v'_1, \ldots, v'_{\mathcal{T}_{\mathcal{L}}}\})$, where $\mathcal{T}_{\mathcal{L}}$ is the reference timeline; $v$ is a value on $\mathcal{T}_{\mathcal{L}}$ which makes the synchronization applicable; $\{\mathcal{T}_{\mathcal{L}_1}, \ldots, \mathcal{T}_{\mathcal{L}_n}\}$ is a set of target timelines on which some values $v'_i$ must hold; and $\mathcal{R}$ is a set of relations which bind temporal occurrence of the reference value $v$ with temporal occurrences of the target values $v'_1, \ldots, v'_{\mathcal{T}_{\mathcal{L}}}$. A plan is defined as a set of timelines $\{\mathcal{T}_{\mathcal{L}_1}, \ldots, \mathcal{T}_{\mathcal{L}_n}\}$ over the same interval for each state variable. A plan is valid with respect to a domain theory if every temporal occurrence of a reference value implies that the related target values hold on target timelines presenting temporal intervals that satisfy the expected relations. A plan is time flexible if $\exists \mathcal{T}_{\mathcal{L}_i} \in \{\mathcal{T}_{\mathcal{L}_1}, \ldots, \mathcal{T}_{\mathcal{L}_n}\}$ such that $\mathcal{T}_{\mathcal{L}_i}$ is time flexible.

At execution time, an executive cannot completely predict the behavior of the controlled physical system because the duration of certain processes or the timing of exogenous events is outside of its control. In these cases, the values for the state variables that are under the executive scope should be chosen so that they do not constrain uncontrollable events. This controllability problem is defined, e.g. in (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 have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) 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 $\mathcal{P}_{\mathcal{L}} = \{\mathcal{T}_{\mathcal{L}_1}, \ldots, \mathcal{T}_{\mathcal{L}_n}\}$, we call projection the set of flexible timelines $\mathcal{P}_{\mathcal{L}'} = \{\mathcal{T}_{\mathcal{L}_1}', \ldots, \mathcal{T}_{\mathcal{L}_n}'\}$ derived from $\mathcal{P}_{\mathcal{L}}$ setting to a fixed value the temporal occurrence of each uncontrollable timepoint. Considering $\mathcal{N}$ as the set of controllable flexible timepoints in $\mathcal{P}_{\mathcal{L}}$, a schedule $T$ is a mapping $T : N \to \mathbb{N}$ where $T(x)$ is called time of timepoint $x$. A schedule is consistent if all value durations and synchronizations are satisfied in $\mathcal{P}_{\mathcal{L}}$. The history of a timeline $x$ w.r.t. a schedule $T$, denoted by $T(<x)$, specifies the time of all uncontrollable timepoints that occur prior to $x$. An execution strategy $S$ is a mapping $S : \mathcal{P} \to T$ where $\mathcal{P}$ is the set of projections and $T$ 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 $\mathcal{P}_{\mathcal{L}}$ is dynamically controllable if there exists a viable execution strategy $S$ such that $S_{p1}(<x) = S_{p2}(<x) \Rightarrow S_{p1}(x) = S_{p2}(x)$ for each controllable timeline $x$ 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. Here, we first present Timed Automata (TA) (Alur and Dill 1994) and then extend them to TGA.

**Basic Definitions**

A fundamental concept in Timed Automata is time. Here, we give the formal definition of clocks and relations that can be defined over them, i.e., how it is possible to model time passing and introduce temporal constraints into automata definition that follows. Formally, we call clock a nonnegative, real-valued variable. Let $X$ be a finite set of clocks. We denote with $C(X)$ the set of constraints $\Phi$ generated by the grammar: $\Phi ::= x \sim c \mid x - y \sim e \mid \Phi \land \Phi$, where $c \in \mathbb{Z}, x,y \in X$, and $\sim \in \{<,\leq,\geq,>\}$. We denote by $B(X)$ the subset of $C(X)$ that uses only the form $x \sim c$. 
Definition 1 A Timed Automaton (TA) (Alur and Dill 1994) is a tuple $A = (Q, q_0, Act, X, Inv, E)$, 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 rectangular constraint $Inv(q)$ (the invariant of $q$), $E \subseteq Q \times B(X) \times Act \times \mathbb{R}^+ \times Q$ is a finite set (or transitions).

In the following, we write $q \xrightarrow{g,a,Y} q' \in E$ for $(g, a, Y, q') \in E$.

A valuation of the variables in $X$ is a mapping from $X$ to the set $\mathbb{R}_{\geq 0}$ of nonnegative reals. We denote with $\mathbb{R}_0^X$ the set of valuations on $X$ and with $\mathbb{0}$ the valuation that assigns the value 0 to each clock. If $Y \subseteq X$ we denote with $v[Y]$ the valuation on $X$ assigning the value $0$ to all clocks in $Y$.

A state of $A$, $q \in Q$, is 4-tuple $(q, g, a, Y)$ where $q \in Q$ is the location, $g, a \in Act$ and $Y \subseteq X$. A state is admissible if $g \models Inv(q)$. A state $(q, g, a, Y)$ is 5-tuple $(q, g, a, Y, v)$ for $v \in \mathbb{R}_0^X$.

A transition $\delta$ of $A$ is a 5-tuple $(q, g, a, Y, v) \xrightarrow{\delta} (q', g', a', Y', v')$ where $q, q' \in Q$, $g, g' \in Act$, $a, a' \in Act$, $Y, Y' \subseteq X$, $v, v' \in \mathbb{R}_0^X$. Transitions are defined in a similar way for TGA.

A time transition is a 4-tuple $(q, g, a, Y) \xrightarrow{\delta} (q', g', a', Y')$, where $q, q' \in Q$, $g, g' \in Act$, $a, a' \in Act$, $Y, Y' \subseteq X$, $v, v' \in \mathbb{R}_0^X$. Transitions are defined in a similar way for TGA.

A run of a TA $A$ is a finite or infinite sequence of alternating time and discrete transitions of $A$. We denote with Runs($A$) the set of runs of $A$ starting from state $(q_0, g_0)$.

A network of TA is a finite set of TA evolving in parallel with a CSS style semantics for parallelism. Formally, let $\mathcal{F} = \{ A_i | i = 1, \ldots, n \}$ be a finite set of automata with $A_i = (Q_i, q_{0i}, Act, X_i, Inv_i, E_i)$ for $i = 1, \ldots, n$. Note that the automata in $\mathcal{F}$ have all the same set of actions and clocks and disjoint sets of locations. The network of $\mathcal{F}$ (notation $|| \mathcal{F}$) is the TA $P = (Q, q_0^P, Act, X, Inv, E)$ defined as follows. The set of locations $Q$ of $P$ 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 $q_0^P$ of $P$ is $q_0^P = (q_{01}, \ldots, q_{0n})$. The invariant Inv of $P$ is Inv($q_1, \ldots, q_n$) = Inv$_1(q_1) \wedge \ldots \wedge$ Inv$_n(q_n)$. The transition relation $E$ for $P$ is the synchronous parallel of those of the automata in $\mathcal{F}$. That is, $E$ consists of the set of $n$-tuples $(q, a, Y, q')$ satisfying the following conditions: 1. $q = (q_1, \ldots, q_n)$, $q' = (q_1', \ldots, q_n')$; 2. There are $i, j \leq n$ 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 $A_i$ of $\mathcal{F}$. 3. Both automata $A_i$ and $A_j$ can make a transition with action $a$. That is, $q_i \xrightarrow{a, Y} q'_i \in E_i$, $q_j \xrightarrow{a, Y} q'_j \in E_j$, $g = g_i \wedge g_j$, $Y = Y_i \cup Y_j$.

Definition 2 A Timed Game Automaton (TGA) is a TA $A = (Q, q_0, Act, X, Inv, E)$ where the set of actions Act is split in two disjoint sets: Act, the set of controllable actions and Act$_u$ the set of uncontrollable actions.

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

Given a TGA $A$ and three symbolic configurations $Init$, $Safe$, and $Goal$, the reachability control problem or reachability game $RG(A, Init, Safe, Goal)$ consists in finding a strategy $f$ such that $A$ starting from $Init$ and supervised by $f$ generates a winning run that stays in $Safe$ and enforces $Goal$. A finite or infinite run $r$ in Runs($A$,$Init$) is winning if either there is some state $(l, v) \in r$ such that $(l, v) \models Goal$ and for all state $(l', v') \in r$ such that $(l', v') \models Safe$. The set of winning runs in $A$ from $Init$ is denoted $WinRuns(Init, A)$. A strategy is a partial mapping $f$ from the set of runs of $A$ starting from $Init$ to the set Act$_u \{\lambda\}$. $f$ is defined in a similar way for TGA.

A state $f$ is state-based or memory-less whenever its result depends only on the last configuration of the run.

Definition 3 Given the TGA $A = (Q, q_0, Act, X, Inv, E)$, a strategy $f$ over $A$ is a partial function from Runs($A$) to Act$_u \{\lambda\}$ s.t. for every finite run $r$, if $f(r) \in Act$, then last($r$) $\xrightarrow{t(o)} (l', v')$ for some $(l', v')$. The restricted behavior of a TGA $A$ controlled with some strategy $f$ is defined by the notion of outcome (de Alfaro, Henzinger, and Majumdar 2001). The outcome Outcome($q, f$) is defined as the subset of Runs($II, A$) that can be generated from $q$ executing the uncontrollable actions in Act$_u$ or the controllable actions provided by the strategy $f$. Focusing on reachability games, a maximal run $r$ is either an infinite run or a finite run that satisfies either i) last($r$) $\models Goal$ or ii) if $r \xrightarrow{a, Y} r'$ then $a \in Act_u$ (i.e. the only possible next discrete actions from last($r$), if any, are uncontrollable actions).

A strategy $f$ is a winning strategy from $q$ if all maximal runs in Outcome($q, f$) are in WinRuns($q, A$). A state $q$ in a TGA $A$ is winning if there exists a winning strategy $f$ from $q$ in $A$. 
UPPAAL-TIGA

This tool (Behrmann et al. 2007) 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.

To model concurrent systems, timed automata can be extended with parallel composition. In the UPPAAL-TIGA modeling language (Larsen, Pettersson, and Yi 1997), the CCS parallel composition operator is used, which allows interleaving of actions as well as handshake synchronization. To model hand-shake synchronization, the action alphabet is assumed to consist of symbols for input action denoted as $a?$, output actions denoted $a!$, and internal actions represented by the distinct symbol $\tau$.

Given a nTGA $\mathcal{N}_A$, a set of goal states ($\text{win}$) and/or a set of bad states ($\text{lose}$), both defined by UPPAAL state formulas, four types of winning conditions can be issued (Behrmann et al. 2007). For all of them, the solution of the game is described by the distinct symbol $\tau$.

To model hand-shake synchronization, the action alphabet is extended with parallel composition. In the UPPAAL-TIGA modeling language (Larsen, Pettersson, and Yi 1997), the action alphabet is extended with parallel composition.

For each planned flexible timeline $T\mathcal{L}$, we define a Timed Game Automaton $A_{T\mathcal{L}} = (Q_{T\mathcal{L}}, q_0, \text{Act}_{T\mathcal{L}}, X_{T\mathcal{L}}, \text{Inv}_{T\mathcal{L}}, E_{T\mathcal{L}})$ as follows:

- for each $i$-th plan step in the flexible plan, we add $l_i$ in $Q_{T\mathcal{L}}$; In addition, a last location $l_{\text{goal}}$ is considered in $Q_{T\mathcal{L}}$;
- $q_0$ is $l_0$;
- for each allowed value $v$ in $SV$, we consider an output action $a_v$; if the related state variable is controllable ( uncontrollable) we add $a_v$ in $\text{Act}_{T\mathcal{L}} (\text{Act}_{aT\mathcal{L}})$;
- we consider the one clock $c_p$ in $X_{T\mathcal{L}}$;
- for each $i$-th plan step and related flexible interval time point $[l_b, l_u]$, we consider $\text{Inv}_{T\mathcal{L}}(l_i) := c_p \leq l_u$; for each $i$-th plan step and related planned value $v_p$, and flexible interval time point $[l_b, l_u]$, we consider a transition $e = q \xrightarrow{g,a,Y} q'$ in $E_{T\mathcal{L}}$, where $q = l_i, q' = l_{i+1}, g = c_p \geq l_b, a = v_p, Y = \emptyset$;
- given the plan length $pl$, we consider a last transition $e = q \xrightarrow{g,a,Y} q'$ in $E_{T\mathcal{L}}$, where $q = l_{pl}, q' = l_{\text{goal}}, g = \emptyset, a = \emptyset, Y = \emptyset$;

The set of automata $\text{Plan} = \{A_{T\mathcal{L}_1}, ..., A_{T\mathcal{L}_n}\}$ constitutes a nTGA that represents the planned timelines description.

State Variables Encoding
For each state variable $SV = (V, T, D)$, we define a Timed Game Automaton $A_{SV} = (Q_{SV}, q_0, \text{Act}_{SV}, X_{SV}, \text{Inv}_{SV}, E_{SV})$ as follows:

- for each allowed value $v$ in $V$, we add a location $l_v$ in $Q_{SV}$;
- $q_0$ is chosen among $Q_{SV}$ elements according to the initial value of the planned flexible timeline on the same state variable $SV$;
- for each allowed value $v$ in $V$, we consider an input action $a_{v?}$; if the state variable is controllable ( uncontrollable) we add $a_{v?}$ in $\text{Act}_{SV} (\text{Act}_{aSV})$;
- we consider one automata clock $c_{SV}$ in $X_{SV}$;
- for each allowed value $v$ in $V$ and $D(v) = [l_b, u_b]$, we define $\text{Inv}_{SV}(v) := c_{SV} \leq u_b$;
- for each allowed value $v$ in $V$, the set of $T(v) = \{v_{s1}, ..., v_{sn}\}$ and the duration constraint $D(v) = [l_b, u_b]$, for each value $v_{si}$ we define a transition $e = q \xrightarrow{g,a,Y} q'$, where $q = l_i, q' = l_{si}$ in $E_{SV}, g = c_{SV} \geq l_b, a = a_{v?}$, $Y = \{c_{SV}\}$.

The set of automata $SV = \{A_{SV_1}, ..., A_{SV_n}\}$ constitutes a nTGA that represents the State Variables description. Note that the use of input and output actions implements the synchronization between state variables and planned timelines. That is, once $A_{T\mathcal{L}}$ fires a transition labeled with $a_{v?}$, the related $A_{SV_i}$ must fire a correspondent transition labeled $a_{v?} (A_{T\mathcal{L}}$, rules $A_{SV_i}$).

Observer Encoding
A last TGA constitutes an Observer automaton that is to supervise the validity of synchronizations and values over $SV$ and $\text{Plan}$.
We define a TGA $A_{Obs} = (Q_{Obs}, q_0, Act_{Obs}, X_{Obs}, Inv_{Obs}, E_{Obs})$ as follows:

- $Q_{Obs} = \{l_{ok}, l_{err}\}$;
- $q_0$ is $l_{ok}$;
- we consider a unique uncontrollable action $a_{fail}$, $Act_{Obs} = Act_{a_{fail}} = \{a_{fail}\}$;
- we consider the same plan clock $X_{Obs} = \{c_p\}$;
- $Inv_{Obs}$ is not defined;
- for each state planned timeline $T_L$ and the related variable $SV$, plan step $s_p$ and related planned value $v_p$, we consider an uncontrollable transition $e = q \xrightarrow{a_{fail}} q’$ in $E_{Obs}$, where $q = l_{ok}$, $q’ = l_{err}$, $g = PT_{sp} \land \neg SV_{vp}$, $l = a_{fail}$, $r = \emptyset$;
- for each synchronization $\langle T_L, v \rangle$ $\rightarrow$ $\langle T_L’1, \ldots, T_L’n, \{v_1', \ldots, v_n'\}, \mathcal{R} \rangle$ we consider an uncontrollable transition $e = q \xrightarrow{a} q’$ in $E_{Obs}$ where $q = l_{ok}$, $q’ = l_{err}$, $g = \neg R(T_L, v, TL_{l_v1}', \ldots, TL_{l_vn})$, $a = a_{fail}$, $Y = \emptyset$.

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

### Verifying Time Flexible Plans

Given the nTGA $PL$ obtained following the execution process presented above, we can define a Reachability Game that ensures, if successfully solved, plan validity.

**Theorem 1** Given a RG($PL, Init, Safe, Goal$) defined considering $Init = \{q \mid q \in Q_{TLi}, \forall TL_i \in Plan\} \cup \{q \mid q \in Q_{SVi}, \forall SV_i \in SV\} \cup \{q \mid q \in Q_{Obs}\}$, $Safe = \{l_{ok}\}$ and $Goal = \{l \mid l \in l_{goal} \in Q_{TL}, \forall TL_i \in Plan\}$, solving/validating the game implies plan validity for $T_L$.

**Proof Sketch.** The proof is composed of two parts. First, we show that the nTGA $PL$ describes all and only the behaviors defined by the flexible plan $P = \{T_L1, \ldots, T_Ln\}$. Then, we prove that solving the $RG(PL, Init, Safe, Goal)$ corresponds to verify the plan.

The set of automata $Plan = \{A_{T_L1}, \ldots, A_{T_Ln}\}$ represents all the possible planned temporal behaviors over all the timelines. In fact, each automaton $A_{TL}$ describes the planned temporal sequence of values for the $TL$ timeline within the planning horizon $H$. While, automata in $SV = \{ASV1, \ldots, ASVn\}$ represent exactly the given state variables description. We recall that the use of input/output actions implements straightforward relations between allowed values and planned values for each timeline. By construction, we have a one-to-one mapping between flexible plan behaviors and automata behaviors: for each behavior in $Plan \cup SV$, we have a behavior in $P$ and vice versa (any possible behavior in $Plan \cup SV$ but not in a flexible plan would violate temporal timepoint plan constraints, any possible flexible plan behavior in $P$ but not in $Plan \cup SV$ would violate automata guards or invariants). Finally, the Observer automaton checks for both values consistency (between planned timelines and state variables) and synchronizations satisfaction. Value consistency is trivial. Again, by construction, the Observer holds into the error location when a transition guard is activated, that is, when the related flexible behavior violates the associated synchronization. On the other hand, when a flexible behavior violates a synchronization, the related guard is activated, hence enforcing the error location for the Observer.

At this point we have that, if there exists a winning strategy $f$ for $RG(PL, Init, Safe, Goal)$, then the Outcome$(Init, f)$ represents the subset of $Runs(PL) \subseteq WinRuns(Init, f)$ that guarantees that (i) Goal states are reached and (ii) Safe states are enforced. This means that each $p \in Outcome(Init, f)$ reaches all the locations in $\{l \mid l is l_{goal} \in Q_{TL}, \forall TL_i \in Plan\}$ while the observer holds $l_{ok}$. From this, it is straightforward that for each timeline $TL_i$, all the transitions can be performed maintaining allowed values (w.r.t. state variable definition) and without violating any synchronization. Thus, the plan is valid.

To search for winning strategies for $RG(PL, Init, Safe, Goal)$ (and then to verify the plan), we exploit UPPAAL-TIGA. This can be done by checking the following formula:

$$\Phi = A [ Safe U Goal].$$

This formula states that, for each possible evolution of uncontrollable state variables, goals must be reached while errors must be avoided. If verified, UPPAAL-TIGA returns a control execution strategy that guarantees (if correctly “executed”) to reach planning goals for all possible temporal world evolutions. Thus, verifying the above property implies validating the flexible temporal plan.

In addition to this, we can ask UPPAAL-TIGA to verify additional properties like, for instance, undesired states avoidance. In fact, Safe configuration can be enriched with additional statements. That is, $Safe = \{l_{ok}\} \cup \{\neg state_{undesired}\}$. Then, the computed strategy ensures not only to reach goals but also to maintain safe state and to avoid undesired states.

Moreover, another important issue can be addressed exploiting our verification approach: plan controllability.

Recalling the dynamic controllability definition for timelines, we notice that: 1) each possible evolution of uncontrollable timeline/automaton in $PL$ corresponds to a projection $p$; 2) each strategy/solution for the $RG$ corresponds to a schedule $T$; 3) a set of winning strategies represents a viable execution strategy $S$.

Thus, UPPAAL-TIGA verifies $\Phi$ (i.e., checks how to win the $RG$) producing a viable execution strategy. Since UPPAAL-TIGA verification process operates on the basis of forward algorithms (Behrmann et al. 2007), the produced execution strategy $S$ is such that $S_{p1}(x) = S_{p2}(x) \Rightarrow S_{p1}(x) = S_{p2}(x)$ for each controllable timepoint $x$ and projections $p1$ and $p2$. As a consequence, we obtain the following Corollary.

**Corollary** Given RG($PL, Init, Safe, Goal$) defined as above and using UPPAAL-TIGA to find a winning execution strategy $S$. If UPPAAL-TIGA solves RG then the flexible plan is dynamically controllable by means of $S$.
We shall notice that our approach to dynamic controllability checking relies on the fact that the verification tool works with forward algorithms; otherwise, nothing can be said about dynamic controllability.

Case Study and Preliminary Experiments

In this section, we present the application of our method in a specific case study. In our recent work, we have considered variants of a real application case studies (Cesta et al. 2008; 2009b). The same experience has been used here to derive a general planning problem. Basically, a remote space agent is to be controlled in order to accomplish some required tasks (science, communication, and maintenance activities). Tasks have to be temporally synchronized with exogenous events that occur independently of agent control.

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

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

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

Any valid plan needs synchronizations among the agent timeline (Figure 2, middle) and the uncontrollable timelines (represented as dotted arrows in Figure 2): science operations must occur during Pericentres, maintenance operations must occur in the same time interval as Apocentres and communications must occur during ground station visibility windows. In addition to those synchronization constraints, the operative mode timeline must respect transition constraints among values and durations for each value specified by the domain (see again Fig. 2).

**Using UPPAAL-TIGA**

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

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

Durations constraints (e.g., Science activity duration in \([2160, 4080]\)) are encoded as both clock mode invariants and guards on the related outgoing transitions. While sequencing constraints (e.g., Science meets Slew) are encoded defining appropriate outgoing transitions.

In Figure 3 we report the complete UPPAAL-TIGA module declaration for the agent state variable.

Plan verification requires an input model that encodes also the generated plan. Since a generated plan provides a set of value activations (associated with time points) (planned timeline) for each state variable, a plan describes the sequence of values the state variables are to assume in a given time frame. To represent flexible plans, we consider an additional general plan clock and we introduce an automaton for each planned behavior. This automaton has a number of modes that equals the length of the plan: for each activation/decision available in the plan we introduce a mode while a final goal mode represents plan completion. An invariant is considered to model maximum staying duration. Transitions between modes represent plan steps, from initial value to the last one. For each transition, we introduce a...
control: A [not monitor .ERR
verify the following formula:
Synchronization constraints among different timelines are
e.g., to check sequencing and synchronizations constraints.
constraints defined both on and among different timelines.
server automaton. It is to check the consistency of temporal
respected, guarantees to reach planning goal in all possible

U plan.Goal
This formula means that for each possible
ence modes.
related transition defined in Figure 3 between Slew and Sci-
are depicted:

Figure 3: Module definition for the Remote Space Agent. Note
that the clock is checked on seconds.
guard that enables transition at the minimum staying dura-
In order to consider both controllable and uncontrollable
state variables, we introduce uncontrollable TGA transitions
for uncontrollable components.

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

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

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

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

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

world evolutions. Thus, verifying the above property im-
plies validating the flexible temporal plan.

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

ing but it also ensures dynamic controllability.

Empirical Results
In order to show the feasibility of our approach, we present
experimental results on preliminary tests focusing on the
analysis of the dependency of plan verification performance
from the degree of flexibility.

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

We perform two kind of experiments. First, keep-
ing constant the plan size (to 35 time points), we study how
plan verification time depends on the number of flexible time

\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]

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_AGTEarthComm) and not (STATIONSAvailable)); }, OK -u-> ERR { guard (REMOTE_AGTMaintenance) and not (ORBIT_EVENTSPericentre)); }, ERR -u-> ERR { }; }

\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]
\[ \Phi = 1 \rightarrow 35, \quad \Delta = 1 \rightarrow 10 \]

process REMOTE_AGT() { state Earth, EarthComm, 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 -> EarthComm { guard clockREMOTE_AGT >= 3600; sync pulse_EarthComm; assign clockREMOTE_AGT := 0; }, EarthComm -> Earth { guard clockREMOTE_AGT >= 0; sync pulse_Earth; assign clockREMOTE_AGT := 0; }, EarthComm -> Maintenance { guard clockREMOTE_AGT >= 0; sync pulse_Maintenance; assign clockREMOTE_AGT := 0; }, EarthComm -> 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 -> EarthComm { guard clockREMOTE_AGT >= 5400; sync pulse_EarthComm; assign clockREMOTE_AGT := 0; }, Slew -> Earth { guard clockREMOTE_AGT >= 1800; sync pulse_Earth; assign clockREMOTE_AGT := 0; }, Slew -> EarthComm { guard clockREMOTE_AGT >= 0; sync pulse_EarthComm; assign clockREMOTE_AGT := 0; }, Slew -> Science { guard clockREMOTE_AGT >= 0; sync pulse_Science; assign clockREMOTE_AGT := 0; } }
points $\Phi$ and on the duration $\Delta$.

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

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

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

Table 2 shows our results for the second kind of experiments. From this figure we see that the verification tool handles well flexible plans with higher and higher degrees of flexibility both in terms of $\Phi$ and $\Delta$.

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

<table>
<thead>
<tr>
<th>$\Phi$</th>
<th>10</th>
<th>20</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35.6±0.8</td>
<td>36.6±1.7</td>
<td>37.4±0.5</td>
</tr>
<tr>
<td>6</td>
<td>35.2±0.4</td>
<td>36 ±0</td>
<td>37.4±0.5</td>
</tr>
<tr>
<td>9</td>
<td>36 ±1.8</td>
<td>36.2±0.4</td>
<td>39.2±1.9</td>
</tr>
<tr>
<td>12</td>
<td>34.8±0.4</td>
<td>36.4±0.5</td>
<td>37.8±0.4</td>
</tr>
<tr>
<td>15</td>
<td>35 ±0</td>
<td>36.2±0.4</td>
<td>43.6±10.2</td>
</tr>
<tr>
<td>18</td>
<td>35 ±0</td>
<td>40±8</td>
<td>39±0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$\Delta$</th>
<th>1</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>40±6</td>
<td>37.4±0.5</td>
<td>37.4±0.4</td>
<td>53.7±8</td>
<td>37.8±1</td>
</tr>
<tr>
<td>6</td>
<td>38.4±0.5</td>
<td>38.6±1.2</td>
<td>38 ±0</td>
<td>44.4±8.5</td>
<td>38.2±0.4</td>
</tr>
<tr>
<td>9</td>
<td>38.4±0.5</td>
<td>38 ±0</td>
<td>39.2±1.9</td>
<td>39±0</td>
<td>38.8±0.4</td>
</tr>
<tr>
<td>12</td>
<td>52.4±10.3</td>
<td>38.8±0.4</td>
<td>38.4±0.5</td>
<td>39±0</td>
<td>39±4±0.5</td>
</tr>
<tr>
<td>15</td>
<td>39.2±0.4</td>
<td>52±13</td>
<td>39.2±0.4</td>
<td>39.2±0.4</td>
<td>39.8±0.4</td>
</tr>
<tr>
<td>18</td>
<td>39.6±0.5</td>
<td>39.6±0.8</td>
<td>40.4±1.5</td>
<td>48.8±9.1</td>
<td>40±0.6</td>
</tr>
</tbody>
</table>

**Conclusion**

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

Notice that, since we use a general purpose model-checker, verification is PSPACE complete. However, this is only a theoretical result and UPPAAL-TIGA algorithm yields very encouraging performance results in practice (Cassez et al. 2005). In fact, the results presented here show that UPPAAL-TIGA allows effective verification of flexible temporal plan by directly using the implicit representation of the state variable models. Therefore, model-checking in UPPAAL-TIGA on the one hand provides a useful independent verification tool for flexible timelines, on the other hand permits plan verification of the flexible plans produced by a black-box planner avoiding to rebuild associated STPU. Moreover, it produces results that can be further exploited as follows. First, from a valid flexible plan we can extract a strategy that can be used to safely execute the given plan. Second, an invalid plan can be analyzed and information can be obtained by the tool, helping users to identify weakness causes and provide useful hints on how to obtain a valid plan.

**Acknowledgments.** Cesta, Fratini, Orlandini and Tronci are partially supported by the EU project ULISSE (Call “SPA-2007.2.1.01 Space Science”, Contract FPT.218815). Cesta and Fratini are also partially supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). Aspects from this paper are synthetically presented in (Cesta et al. 2009a).

**References**


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 9th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space. JPL, Pasadena, CA.


