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 2000). 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 a *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 $\langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$ where: (a) $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *values*; (b) $\mathcal{T} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$ is the *value transition* function; (c) $\mathcal{D} : \mathcal{V} \rightarrow \mathbb{N} \times \mathbb{N}$ is the *value duration* function, i.e. a function that specifies the allowed duration of values in $\mathcal{V}$ (as an interval $[lb, ub]$). Given a state variable, its associated *timeline* is represented as a sequence of values in the temporal interval $\mathcal{H} = [0, H)$. Each value satisfies previous (a-b-c) specifications and is defined on a set of not overlapping time intervals contained in $\mathcal{H}$. We suppose that adjacent intervals present different values. A timeline is said *completely specified* over the temporal horizon $\mathcal{H}$ when a sequence of non-overlapping valued intervals exists and its union is equal to $\mathcal{H}$. A timeline is said *time-flexible* when is completely specified and transition events are associated to temporal intervals (lower and upper bounds are given for them), instead of exact temporal occurrences. In other words, a time-flexible timeline represents a set of timelines, all sharing the same sequence of values. It is worth noting that not all the timelines in this set are valid (satisfies a-b-c). The process of *timeline extraction* from a time-flexible timeline is the process of computing (if exists) a valid and completely specified timeline from a given time-flexible timeline. In timeline-based planning, a *planning domain* is defined as a set of state variables $\{\mathcal{SV}_1, \ldots, \mathcal{SV}_n\}$ that cannot be considered as reciprocally decoupled. Then, a *domain theory* is defined as a set of additional relations, called *synchronizations*, that model the existing temporal constraints among state variables. A synchronization has the form $\langle \mathcal{TL}, v \rangle \longrightarrow \langle \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}, \{v'_1, \ldots, v'_{|\mathcal{TL}'|}\}, \mathcal{R} \rangle$ where: $\mathcal{TL}$ is the reference timeline; v is a value on $\mathcal{TL}$ which makes the synchronization applicable; $\{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}$ is a set of target timelines on which some values v'_j must hold; and $\mathcal{R}$ is a set of *relations* which bind temporal occurrence of the *reference* value v with temporal occurrences of the *target* values $v'_1, \ldots, v'_{|\mathcal{TL}'|}$. A *plan* is defined as a set of timelines $\{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ over the same interval for each state variable. A plan is *valid* with respect to a domain theory if every temporal occurrence of a reference value implies that the related target values hold on target timelines presenting temporal intervals that satisfy the expected relations. A plan is *time flexible* if $\exists \mathcal{TL}_i \in \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$ such that $\mathcal{TL}_i$ is time flexible.

At execution time, an executive cannot completely predict the behavior of the controlled physical system because the duration of certain processes or the timing of exogenous events is outside of its control. In these cases, the values for the state variables that are under the executive scope should be chosen so that they do not constrain uncontrollable events. This *controllability problem* is defined, e.g. in (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 $\mathcal{PL} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$, 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. 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 Φ generated by the grammar: $\Phi ::= x \sim c \mid x - y \sim c \mid \Phi \wedge \Phi$, where $c \in Z$, $x, y \in X$, and $\sim \in \{<, \leq, \geq, >\}$. We denote by $B(X)$ the subset of $C(X)$ that uses only the form $x \sim c$.

Definition 1 *A* **Timed Automaton (TA)** *(Alur and Dill 1994) 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*),* X *is a finite set of clocks,* $\mathrm{Inv} : Q \to B(X)$ *is a function associating to each location* $q \in Q$ *a rectangular 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 (or* transitions*).*

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

A *valuation* of the variables in X is a mapping v from X to the set $R_{\geq 0}$ of nonnegative reals. We denote with $R^X_{\geq 0}$ the set of valuations on X and with $\vec{0}$ the valuation that assigns the value 0 to each clock. If $Y \subseteq X$ we denote with $v[Y]$ the valuation (on X) assigning the value 0 ($v(z)$) to any $z \in Y$ ($z \in (X - Y)$). For any $\delta \in R^{\geq 0}$ we denote with $(v + \delta)$ the valuation such that, for each $x \in X$, $(v+\delta)(x) = v(x) + \delta$. Let $g \in C(X)$ and v be a valuation. We say that g satisfies v, notation $v \models g$ if constraint g evaluated on v returns true. This basic model of TA can be extended to allow location variables with finite values in guards, invariants, and assignments.

A *state* of TA $\mathcal{A} = (Q, q_0, \mathrm{Act}, X, \mathrm{Inv}, E)$ is a pair (q, v) s.t. $q \in Q$ and v is a valuation (on X). We denote with S the set of states of $\mathcal{A}$. An *admissible* state for a $\mathcal{A}$ is a state (q, v) s.t. $v \models \mathrm{Inv}(q)$.

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

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

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

Given a TGA $\mathcal{A}$ and three symbolic configurations *Init*, *Safe*, and *Goal*, the *reachability control problem* or reachability game $RG(\mathcal{A}$, *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 finite or infinite run ρ in Runs($\mathcal{A}$,*Init*) is *winning* if either there is some state $(l, v) \in \rho$ such that $(l, v) \models$ *Goal* and for all state $(l', v') \in \rho$ $(l', v') \models$ *Safe*. The set of winning runs in $\mathcal{A}$ from *Init* is denoted *WinRuns(Init*, $\mathcal{A}$). 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 $f(\rho) = l$. A strategy f is *state-based* or *memory-less* whenever its result depends only on the last configuration of the run.

Definition 3 *Given the TGA* $\mathcal{A} = (Q, q_0, \mathrm{Act}, X, \mathrm{Inv}, E)$*, a* **strategy** f *over* $\mathcal{A}$ *is a partial function from* $Runs(\mathcal{A})$ *to* $Act_c \cup \{\lambda\}$ *s.t. for every finite run* ρ*, if* $f(\rho) \in Act_c$ *then* $last(\rho) \overset{f(\rho)}{\to} (l', v')$ *for some* (l', v')*.*

The restricted behavior of a TGA $\mathcal{A}$ 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 $Runs(\Pi, \mathcal{A})$ 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* ρ is either an infinite run or a finite run that satisfies either i) $last(\rho) \models Goal$ or ii) if $\rho \overset{a}{\to}$ then $a \in Act_u$ (i.e. the only possible next discrete actions from $last(\rho)$, if any, are uncontrollable actions).

A strategy f is a *winning strategy* from q if all maximal runs in $Outcome(q, f)$ are in $WinRuns(q, \mathcal{A})$. A state q in a TGA $\mathcal{A}$ is *winning* if there exists a winning strategy f from q in $\mathcal{A}$.

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 hand-shake 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 τ.

Given a nTGA $\mathcal{N}_A$, a set of goal states (*win*) and/or a set of bad states (*lose*), both defined by UPPAAL state formulas, four types of winning conditions can be issued (Behrmann et al. 2007). For all of them, the solution of the game is a controllable strategy f such that $\mathcal{N}_A$ supervised by f ensures that: $A\Diamond\, win$ (must reach win); $A[\,not(lose)\,U\,win]$ (must reach win and must avoid lose); $A[\,not(lose)\,W\,win]$ (should reach win and must avoid lose); $A\Box\, not(lose)$ (must avoid lose).

Building TGA from Timeline-based Planning Specifications

The main contribution of this work is in showing how flexible timeline-based plan verification can be performed solving a TGA Reachability Game. To this end, this section describes how we encode a flexible timeline-based plan and the related domain theory into a suitable nTGA. While, the next section presents the Reachability Game definition and how UPPAAL-TIGA can be exploited to solve it.

Concerning the encoding, we first define a TGA for each planned flexible timeline. Then, for each state variable $\mathcal{SV}$, we define a correspondent TGA, while Domain Theory is encoded by means of an Observer automaton. This also checks that plan and state variables assume consistent values over all the planning horizon $\mathcal{H}$. In general, state variables can present both controllable and uncontrollable values. Here, we choose to partition state variables in controllable and uncontrollable, simplifying the formalization. But, it is worth noting that we are able to handle the general case as well.

Flexible Plan Encoding

Given a flexible plan $\mathcal{P} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$, we define a TGA for each $\mathcal{TL}_i$. We consider a unique overall *plan clock* c_p. Each automaton has the same number of states as the length of the timeline: for each activation available in the plan we introduce a state while a final *goal* state represents plan completion.

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

- for each i-th plan step in the flexible plan, we add l_i in $Q_{\mathcal{TL}}$; In addition, a last location l_{goal} is considered in $Q_{\mathcal{TL}}$;
- q_0 is l_0;
- for each allowed value v in $\mathcal{SV}$, we consider an output action $a_v!$; if the related state variable is controllable (uncontrollable) we add a_v in $\text{Act}_{c\mathcal{TL}}$ ($\text{Act}_{u\mathcal{TL}}$);
- we consider the one clock c_p in $X_{\mathcal{TL}}$;
- for each i-th plan step and related flexible interval time point $[lb, ub]$, we consider $\text{Inv}_{\mathcal{TL}}(l_i) := c_p \leq ub$;
- for each i-th plan step and related planned value v_p and flexible interval time point $[lb, ub]$, we consider a transition $e = q \overset{g,a,Y}{\to} q'$ in $E_{\mathcal{TL}}$, where $q = l_i$, $q' = l_{i+1}$, $g = c_p \geq lb$, $a = v_p!$, $Y = \emptyset$;
- given the plan length pl, we consider a last transition $e = q \overset{g,a,Y}{\to} q'$ in $E_{\mathcal{TL}}$, where $q = l_{pl}$, $q' = l_{goal}$, $g = \emptyset$, $a = \emptyset$, $Y = \emptyset$;

The set of automata $Plan = \{\mathcal{A}_{\mathcal{TL}_1}, ..., \mathcal{A}_{\mathcal{TL}_n}\}$ constitutes a nTGA that represents the planned timelines description.

State Variables Encoding

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

- for each allowed value v in $\mathcal{V}$, 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 $\mathcal{SV}$;
- for each allowed value v in $\mathcal{V}$, we consider an input action $a_v?$; if the state variable is controllable (uncontrollable) we add a_v in Act_{cSV} (Act_{uSV});
- we consider one automata clock c_{sv} in X_{SV};
- for each allowed value v in $\mathcal{V}$ and $\mathcal{D}(v) = [l_b, u_b]$, we define $\text{Inv}_{SV}(v) := c_{sv} \leq u_b$;
- for each allowed value v in $\mathcal{V}$, the set of $\mathcal{T}(v) = \{vs_1, ..., vs_n\}$ and the duration constraint $\mathcal{D}(v) = [l_b, u_b]$, for each value vs_i we define a transition $e = q \overset{g,a,Y}{\to} q'$, where $q = l_v$, $q' = l_{vs_i}$ in E_{SV}, $g = c_{SV} \geq l_b$, $a = a_{q'}?$, $Y = \{c_{SV}\}$.

The set of automata $SV = \{\mathcal{A}_{SV_1}, ..., \mathcal{A}_{SV_n}\}$ 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 $\mathcal{A}_{\mathcal{TL}_i}$ fires a transition labeled with $a_v!$, the related $\mathcal{A}_{SV_1}$ must fire a correspondent transition labeled $a_v?$ ($\mathcal{A}_{\mathcal{TL}_i}$ rules $\mathcal{A}_{SV_i}$).

Observer Encoding

A last TGA constitutes an *Observer* automaton that is to supervise the validity of synchronizations and values over SV and $Plan$.

We define a TGA $\mathcal{A}_{Obs} = (Q_{Obs}, q_0, \text{Act}_{Obs}, X_{Obs}, \text{Inv}_{Obs}, E_{Obs})$ as follows:

- $Q_{Obs} = \{l_{ok}, l_{err}\}$;
- q_0 is l_{ok};
- we consider a unique uncontrollable action a_{fail}, $\text{Act}_{Obs} = \text{Act}_{uObs} = \{a_{fail}\}$;
- we consider the same plan clock $X_{Obs} = \{c_p\}$;
- Inv_{Obs} is not defined;
- for each state planned timeline $\mathcal{TL}$ and the related variable $\mathcal{SV}$, plan step s_p and related planned value v_p, we consider an uncontrollable transition $e = q \overset{g,l,r}{\rightarrow} q'$ in E_{Obs}, where $q = l_{ok}$, $q' = l_{err}$, $g = \mathcal{PT}_{s_p} \wedge \neg SV_{v_p}$, $l = a_{fail}$, $r = \emptyset$;
- for each synchronization $\langle \mathcal{TL}, v \rangle \longrightarrow \langle \{\mathcal{TL}'_1, \ldots, \mathcal{TL}'_n\}, \{v'_1, \ldots, v'_n\}, \mathcal{R} \rangle$ we consider an uncontrollable transition $e = q \overset{g,a,Y}{\rightarrow} q'$ in E_{Obs} where $q = l_{ok}$, $q' = l_{err}$, $g = \neg\mathcal{R}(\mathcal{TL}_v, \mathcal{TL}'_{1_{v'_1}}, \ldots, \mathcal{TL}'_{n_{v'_n}})$, $a = a_{fail}$, $Y = \emptyset$.

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

Verifying Time Flexible Plans

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

Theorem 1 *Given a $RG(\mathcal{PL}$,* Init, Safe, Goal*) defined considering* Init = $\{q \mid q \; is \; q_0 \in Q_{\mathcal{TL}_i} \; \forall \mathcal{TL}_i \in Plan\} \cup \{q \mid q \; is \; q_0 \in Q_{\mathcal{SV}_i} \; \forall \mathcal{SV}_i \in SV\} \cup \{q \mid q \; is \; q_0 \in Q_{Obs}\}$, Safe = $\{l_{ok}\}$ *and* Goal = $\{l \mid l \; is \; l_{goal} \in Q_{\mathcal{TL}_i} \; \forall \mathcal{TL}_i \in Plan\}$*, solving/winning the game implies plan validity for $\mathcal{TL}$.*

Proof Sketch. The proof is composed of two parts. First, we show that the nTGA $\mathcal{PL}$ describes all and only the behaviors defined by the flexible plan $\mathcal{P} = \{\mathcal{TL}_1, \ldots, \mathcal{TL}_n\}$. Then, we prove that solving the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) corresponds to verify the plan.

The set of automata $Plan = \{\mathcal{A}_{\mathcal{TL}_1}, ..., \mathcal{A}_{\mathcal{TL}_n}\}$ represents all the possible planned temporal behaviors over all the timelines. In fact, each automaton $\mathcal{A}_{\mathcal{TL}_i}$ describes the planned temporal sequence of values for the $\mathcal{TL}_i$ timeline within the planning horizon $\mathcal{H}$. While, automata in *SV* = $\{\mathcal{A}_{SV_1}, ..., \mathcal{A}_{SV_n}\}$ 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 $\mathcal{P}$ 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 $\mathcal{P}$ 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 $RG(\mathcal{PL}, Init, Safe, Goal)$, then the $Outcome(Init, f)$ represents the subset of $Runs(\mathcal{PL}) \subseteq$ *WinRuns*$(Init, f)$ that guarantees that (i) $Goal$ states are reached and (ii) $Safe$ states are enforced. This means that each $\rho \in Outcome(Init, f)$ reaches all the locations in $\{l \mid l \; is \; l_{goal} \in Q_{\mathcal{TL}_i} \; \forall \mathcal{TL}_i \in Plan\}$ while the observer holds l_{ok}. From this, it is straightforward that for each timeline $\mathcal{TL}_i$, 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 $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) (and then to verify the plan), we exploit UPPAAL-TIGA. This can be done by checking the following formula: Φ = *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 $\mathcal{PL}$ 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 Φ (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 $S_{p1}\{\prec x\} = S_{p2}\{\prec x\} \Rightarrow S_{p1}(x) = S_{p2}(x)$ for each controllable timepoint x and projections $p1$ and $p2$. As a consequence, we obtain the following Corollary.

Corollary 1 *Given $RG(\mathcal{PL}$,* 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 from agent control.

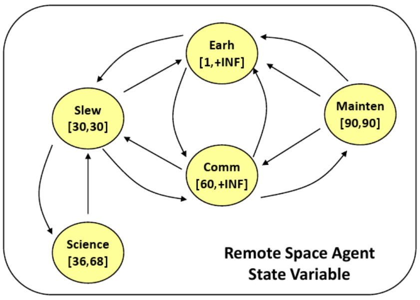

Figure 1: Value transitions for the a main state variable describing the Remote Space Agent temporal behavior.

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

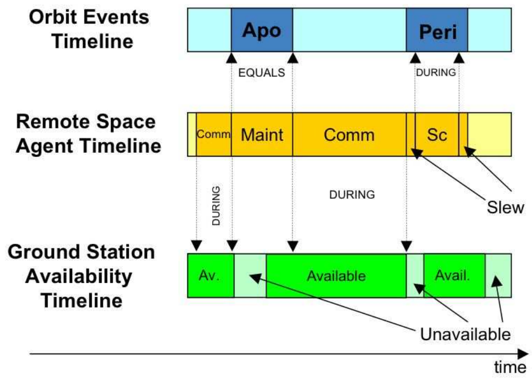

Figure 2: Timeline synchronizations in a plan.

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

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

Figure 3: Module definition for the Remote Space Agent. Note that the clock is checked on seconds.

guard that enables transition at the minimum staying duration.

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

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

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

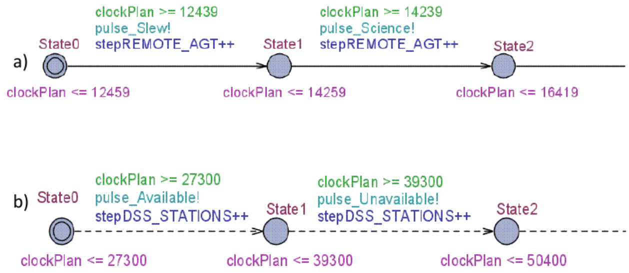

Figure 4: TIGA models for timelines: a) controllable state variable; b) uncontrollable state variable.

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

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

world evolutions. Thus, verifying the above property implies validating the flexible temporal plan.

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

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 parameters we consider are: the number Φ of time points that are replaced with time intervals and the width (*duration*) Δ of such intervals.

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

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

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

Table 1 shows our results for the first kind of experiments. From this figure we see that the verification tool 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 plan with higher and higher degrees of flexibility both in terms of Φ and Δ.

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

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

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

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

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 FP7.218815). Cesta and Fratini are also partially supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). Aspects from this paper are synthetically presented in (Cesta et al. 2009a).

References

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

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

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

Bresina, J.; Jonsson, A.; Morris, P.; and Rajan, K. 2004. Mixed-initiative constraint-based activity planning for mars exploration rovers. In *IWPSS-04. Proceedings of 4th International Workshop on Planning and Scheduling for Space*.

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

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

Cesta, A.; 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. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review (To appear)*.

de Alfaro, L.; Henzinger, T.; and Majumdar, R. 2001. Symbolic Algorithms for Infinite-state Games. In *Proceedings of the 12th International Conference on Concurrency Theory*, pp. 536–550. Lecture Notes in Computer Science 2154, Springer-Verlag.

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

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

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

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

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.

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

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

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

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

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