USING VALIDATION AND VERIFICATION TECHNIQUES FOR ROBUST PLAN EXECUTION

Andrea Orlandini[1], Alberto Finzi[2], and Amedeo Cesta[1]

[1]*CNR – Consiglio Nazionale delle Ricerche, ISTC, Rome, Italy – name.surname@istc.cnr.it*
[2]*DSF – Università Federico II, Naples, Italy – finzi@na.infn.it*

ABSTRACT

This paper describes the exploitation of a Validation and Verification technique aiming at enriching the support capabilities of the KnowledgE ENgineering (KEEN) software environment. In particular, the work reports on the formal synthesis of a plan controller associated to a flexible temporal plan. The controller synthesis exploits Timed Game Automata (TGA) for formal modeling and UPPAAL-TIGA as a model checker. The paper introduces a detailed experimental analysis on a real-world case study demonstrating the viability of the approach. In particular, it is shown how the controller synthesis overhead is compatible with the performance expected from a short horizon planner.

Key words: Design Support System, Validation and Verification, Timeline-based Planning and Execution.

1. INTRODUCTION

Space missions are very demanding in terms of both requirements and costs. Ensuring the validity of plans is of key importance for accepting automated planning technology in the loop of space missions. For this reason these authors are working at integrating Verification and Validation (V&V) with plan generation tools [10]. In particular to enhance the reliability and maintainability of tools supporting plan generation, we are working at the incremental synthesis of the KnowledgE ENgineering (KEEN) environment [8]. The KEEN system has been shown as well suited for supporting design and development of timeline-based Planning and Scheduling (P&S) applications in space mission operations. A particular strong requirement for tools supporting space missions is related to robust plan execution in uncertain and dynamic environments, which is a critical issue, for example, for plan-based autonomous systems. In fact, once a planner has synthesized a temporal plan, it is up to the executive system to decide, at run-time, how and when to exactly execute each planned activity preserving both plan consistency and controllability. Such a capability is even more crucial when the generated plan is temporally flexible being such a plan only partially specified.

Previous works have tackled these issues within a Constraint-based Temporal Planning (CBTP) framework deploying specialized techniques based on temporal-constraint networks. Several authors [19, 18, 23] proposed a *dispatchable execution* approach where a flexible temporal plan is then used by a plan executive that schedules activities on-line while guaranteeing constraint satisfaction.

In our recent work [21], the robust plan execution is pursued relying on Timed Game Automata (TGA) formal modeling and controller synthesis. Recently, we have integrated such an approach in the KEEN system that now is endowed with an alternative and novel approach to flexible plan dispatching/execution. The technique used to synthesize plan controllers is a direct consequence of the formalization proposed in [9]. Analogously to that work, the generated flexible temporal plan and the dynamic domain are encoded into TGA models. However, a different perspective is investigated by exploiting a model checker (i.e., UPPAAL TIGA) to directly synthesize a real-time plan controller for the flexible plan. Such controller guarantees plan execution along with other domain dependent properties.

This paper offers a synthesis of the recent developments of the KEEN environment, in particular showing the achievements in supporting plan execution.

2. PLAN-BASED ROBOT CONTROL

We started our work on the integration of V&V and P&S within the EU Project ULISSE [1] while we have addressed plan-based autonomy within the GOAC project [2]. We are currently pursuing the KEEN development as an internal initiative of our group in the aim of integrating different capabilities to support new research. In particular we are focusing on capabilities for robust plan execution in robots.

The Goal Oriented Autonomous Controller [5] is an ESA effort to create a common platform for robotic software development. In particular, the delivered GOAC architecture has integrated: (a) a timeline-based deliberative layer which integrates a planner based on the APSI Platform [12] and an executive a la T-REX [22]; (b) a functional layer which integrates G$^{en}{}_{o}$M and BIP [2]. In the paper, we use a real-world running example taken from the

[1] http://www.ulisse-space.eu/

[2] ESA Contract TRP/T313/006MM

GOAC project. However, independent on our current use, the work described in this paper is valid for any generic layered control architecture (e.g., [15]) that integrates a temporal planning and scheduling system.

The Robotic Domain. Let us consider a planetary rover equipped with a Pan-Tilt Unit (PTU), two stereo cameras (mounted on top of the PTU) and a communication facility. The rover is able to autonomously navigate the environment, move the PTU, take pictures and communicate images to a Remote Orbiter. Finally, during the mission, the Orbiter may be not visible for some periods. Thus, the robotic platform can communicate only when the Orbiter is visible. The mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration. A possible mission action sequence is the following: navigate to one of the requested locations, move the PTU pointing at the requested direction, take a picture, then, communicate the image to the orbiter during the next available visibility window, put back the PTU in the safe position and, finally, move to the following requested location. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed. The rover must operate following some operative rules to maintain safe and effective configurations. Namely, the following conditions must hold during the overall mission: **(C1)** While the robot is moving the PTU must be in the safe position (pan and tilt at 0); **(C2)** The robotic platform can take a picture only if the robot is still in one of the requested locations while the PTU is pointing at the related direction; **(C3)** Once a picture has been taken, the rover has to communicate the picture to the base station; **(C4)** While communicating, the rover has to be still; **(C5)** While communicating, the orbiter has to be visible.

3. TIMELINE-BASED PLANNING AND EXECUTION

Timeline-based planning is an approach to temporal planning that has been applied in the solution of several real world problems – e.g., [20]. The approach pursues a general idea that planning and scheduling for controlling complex physical systems consists in the synthesis of desired temporal behaviors (or *timelines*).

State variables and timelines. According to this paradigm a domain is modeled as a set of features with an associated set of temporal functions on a finite set of values. The time varying features are called *multi-valued state variables* as in [20]. As in classical control theory, the evolution of the features is described by some causal laws and limited by domain constraints. These are specified in a *domain specification*. The task of a planner is to find a sequence of decisions that brings the timelines into a final desired set always satisfying the domain specification and special conditions called *goals*. We assume that the temporal features have a finite set of possible values assumed over temporal intervals. The temporal evolutions are sequences of operational states. Causal and temporal constraints specify which value transitions are allowed, the duration of each valued interval and synchronization constraints between different state variables. More formally, a state variable is defined by a tuple $\langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$ where: (a) $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *values*; (b) $\mathcal{T} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$ is the *value transition* function; (c) $\mathcal{D} : \mathcal{V} \rightarrow \mathbb{N} \times \mathbb{N}$ is the *value duration* function, i.e. a function that specifies the allowed duration of values in $\mathcal{V}$ (as an interval $[lb, ub]$). (b) and (c) specify the operational constraints on the values in (a). 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 $\mathcal{H}$.

Timeline specification for the robotic domain. To obtain a timeline-based specification of our robotic domain, we consider two types of state variables: *Planned State Variables* to represent timelines whose values are decided by the planning agent, and *External State Variables* to represent timelines whose values over time can only be observed. Planned state variables are those representing time varying features like the temporal occurrence of navigation, PTU, camera and communication operations. We use four of such state variables, namely the *RobotBase*, *PTU*, *Camera* and *Communication*.

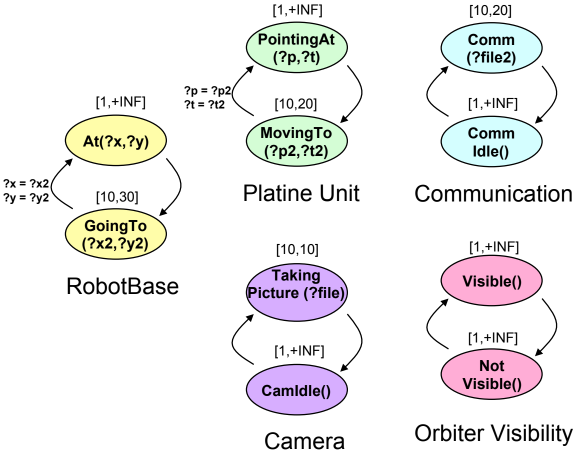

Figure 1. State variables describing the robotic platform and the orbiter visibility (durations are stated in seconds)

In Fig. 1, we detail the values that can be assumed by these state variables, their durations and the legal value transitions in accordance with the mission requirements and the robot physics[3]. Additionally, one external state variable represents contingent events, i.e., the communication opportunities. The *Orbiter Visibility* state variable maintains the visibility of the orbiter. The allowed values for this state variable is *Visible* or *Not-Visible* and are set as an external input. The robot can be in a position (*At(x,y)*) or moving towards a destination (*GoingTo(x,y)*). The PTU can assume a *PointingAt(pan,tilt)* value if pointing a certain direction, while, when moving, it assumes a *MovingTo(pan,tilt)*. The camera can take a picture of a given object in a position $\langle x, y \rangle$ with the PTU in $\langle pan, tilt \rangle$ and store it as a file in the on-board memory (*TakingPicture(file-id,x,y,pan,tilt)*) or be idle (*CamIdle()*). Similarly, the communication facility can be operative and dumping a given file (*Communicating(file-id)*) or be idle (*ComIdle()*).

[3] Note that variables (e.g., ?x) represents parameters with values in a finite set of symbols, used to compactly represent the allowed values for a given state variable. Moreover, the symbol +INF is used to state the upper bound of the temporal interval [0,H).

Representing domain causality. Domain operational constraints are described by means of *synchronizations*. A synchronization models the existing temporal and causal constraints among the values taken by different timelines (i.e., patterns of legal occurrences of the operational states across the timelines).

Fig. 2 exemplifies the use of synchronizations implementing the operative rules (see Section 2) in our case study domain. The synchronizations depicted are: *GoingTo(x,y)* must occur during *PointingAt(0,0)* (**C1**); *TakingPicture(pic,x,y,pan,tilt)* must occur during *At(x,y)* and *PointingAt(pan,tilt)* (**C2**); *TakingPicture(pic,x,y,pan,tilt)* must occur before *Communicating(pic)* (**C3**); *Communicating(file)* must occur during *At(x,y)* (**C4**); *Communicating(file)* must occur during *Visible* (**C5**).

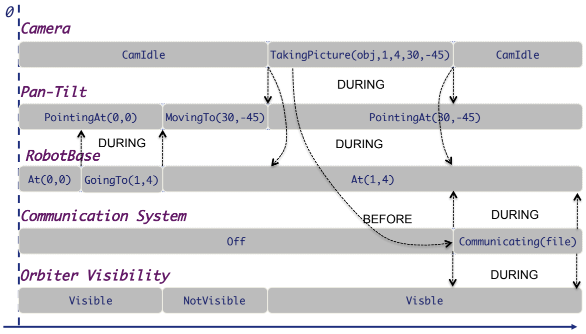

Figure 2. An example of timeline-based plan with synchronizations.

In addition to those synchronization constraints, the timelines must respect transition constraints among values and durations for each value specified in the domain (see again Fig. 1).

Timeline-based planning. *Planning goals* are expressed as desired timeline values in temporal intervals; the planning task is to build a set of timelines that describe valid sequences of values that achieve the desiderata. Hence, a *plan* is a set of *timelines*, that is, a sequence of state variable values, a set of ordered transition points between the values, and a set of distance constraints between transition points. When the transition points are bounded by the planning process (lower and upper bounds are given for them) instead of being exactly specified, we refer to the timeline as *time flexible* while a *flexible plan* is the plan resulting from a set of flexible timelines. A flexible plan defines a set of admissible temporal behaviors. Considering a partial horizon H' (with $H' < H$), the same flexible plan defines a set of *partial* temporal behaviors $\mathcal{PB}$. A *flexible plan* $\mathcal{P} = \{TL_1, ..., TL_n\}$ is defined over a given horizon $\mathcal{H}$. A solution plan is *valid* with respect to a domain theory if every temporal occurrence of a value satisfies the expected synchronizations.

Plan execution. During plan execution, the plan, or a partial segment of it, is under responsibility of the executive system that forces value transitions over the timelines dispatching commands to the functional layers while continuously accepting observations and, thus, monitoring the plan execution. Additionally, not all the value transitions are under responsibility of the executive, but events exist that are under control of the *environment*. In such cases, the values for the controllable state variables should be chosen so that they do not constrain uncontrollable events. This is the *controllability problem* ([24]). Controllability issues underlying a plan representation have been formalized and investigated for the Simple Temporal Problems with Uncertainty (STPU) representation in [24] where basic formal notions are given for *dynamic* controllability (see also [18]). In [9] these notions have been extended to the timeline-based framework.

In order to endow the executive system with an execution strategy, a plan controller is needed taking into account also the controllability problem. That is, the executive system is to robustly execute a flexible temporal plan. More formally, a *plan controller* $\mathcal{C}$ is a partial function from the set of partial behaviors $\mathcal{PB}$ and possible horizons to the set of controllable values for state variables plus a special action λ representing the *wait* action, $\mathcal{C} : \mathcal{PB} \times \mathbb{N} \rightarrow \mathcal{V}_1 \cup ... \cup \mathcal{V}_n \cup \{\lambda\}$.

4. THE KEEN ENVIRONMENT

Validation and Verification techniques may represent a complementary technology, with respect to P&S, and can be used to obtain richer software development environments able to synthesize a new generation of robust problem-solving applications [10]. In fact, developing a P&S application requires several design phases and V&V support tools can alleviate the work of knowledge engineers deputed to build such applications. In particular we are here considering the perspective of a ground segment mission environment within which plans for a remote device are prepared and tested before upload. After quite an amount of work in developing specific applications for ESA (e.g., [6, 3, 7]) and in creating the APSI-TRF infrastructure for P&S [12] in more recent work we have dedicated attention to knowledge engineering supports for different users and in particular on the issue of validating the work of a planner developed within such an infrastructure. Our starting point has been to re-create within the APSI-TRF structured style of implementation a general purpose timeline planner like OMPS [14]. The use of the APSI-TRF is shown in the central block of Figure 3: the Component-Based Modeling Engine made available by the TRF coupled with a search engine (called generically Problem Solver in the figure) developed on purpose creates a complete planner able to synthesize timeline-based solutions for planning problems. Additional flexibility is offered by two input languages (the Domain Description Language and Domain Description Language) that offer flexibility to the use of the tool. Finally, a Plan Execution block has been considered. This is based on a Dispatch Service to send control commands to the controlled system (e.g., a robot) and an Execution Feedback module that allows to receive the telemetry from actual plan execution on physical system. A possible instance of the Plan Execution layer is the one described in [13] that can be easily tailored for different ground segment needs. Around this core tool we have built several engineering services based on our recent work on V&V issues.

In [8], the KnowledgE ENgineering (KEEN) design support prototype system has been presented. It is composed of different V&V modules implementing different design support functionalities (see Figure 3).

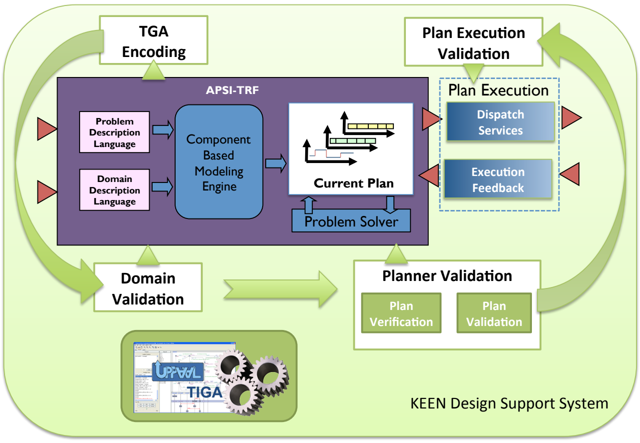

Figure 3. The KnowledgE ENgineering (KEEN) Design Support System.

A *TGA Encoding* module is deputed to implement a translation from P&S specification to TGA. The encoding method is the same presented in [9] allowing to share the same formal results presented in [11]. The other modules rely on that encoding. A *Domain Validation* module is to support the model building activity providing a tool to assess the quality of the P&S models. A *Planner Validation* module is deputed to assess the P&S solver with respect to system requirements. In this regard, two sub-modules are needed: *Plan Verification* to verify the correctness of solution plans and *Plan Validation* to evaluate their goodness. Then, a *Plan Execution Validation* module is to check whether proposed solution plans are suitable for actual execution or not. To implement the modules functionalities, verification tasks are performed by means of UPPAAL-TIGA [1]. This tool extends UPPAAL [16] providing a toolbox for the specification, simulation, and verification of real-time games. As a result, UPPAAL-TIGA is the core engine of the KEEN design support system.

5. ENRICHING KEEN WITH ROBUST PLAN CONTROLLER GENERATION CAPABILITY

Timed Game Automata [17] (TGA)allow to model real-time systems and controllability problems representing uncontrollable activities as *adversary moves* within a game between the controller and the environment. Following the same approach presented in [9] (and briefly discussed above), flexible timeline-based plan verification can be performed by solving a Reachability Game using UPPAAL-TIGA. To this end, we compile flexible timeline-based plans, state variables, and domain theory descriptions into a set of TGA (nTGA). This is obtained with the following steps: (1) a flexible timeline-based plan $\mathcal{P}$ is mapped into a nTGA *Plan*. Each timeline is encoded as a sequence of locations (one for each timed interval), while transition guards and location invariants are defined according to (respectively) lower and upper bounds of flexible timed intervals; (2) the set of state variables *SV* is mapped into a nTGA *StateVar*. Basically, we define a one-to-one mapping from state variables descriptions to TGA. In this encoding, value transitions are partitioned into controllable and uncontrollable. (3) an *Observer* automaton is introduced to check for value constraints violations and synchronizations violations. In particular, we have two locations: an Error location, to state constraint/synchronization violations, and a Nominal (OK) location, to state that the plan behavior is correct. The *Observer* is defined as fully uncontrollable. (4) the nTGA $\mathcal{PL}$ composed by the set of automata *StateVar* $\cup$ *Plan* $\cup$ $\{\mathcal{A}_{Obs}\}$ encapsulates flexible plan, state variables and domain theory descriptions.

Considering a Reachability Game $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) where *Init* represents the set of the initial locations of each automaton in $\mathcal{PL}$, *Safe* is the Observer's OK location, and *Goal* is the set of goal locations, one for each automaton in *Plan*, plan verification can be performed solving the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) defined above. If there is no winning strategy, UPPAAL-TIGA provides a counter strategy for the opponent (i.e., the environment) to make the controller lose. That is, an execution trace showing a faulty evolution of the plan is provided.

5.1. Generating Controllers for Flexible Plan Execution: a TGA approach

Recently, the KEEN environment has been endowed with a new functionality: a method to synthesize robust plan controllers for timeline-based flexible plans solving a TGA model checking problem [21]. Then, a one-to-one mapping between flexible temporal behaviors over $[0, H]$ defined by $\mathcal{P}$ and the automata behaviors defined by $\mathcal{PL}$ is presented. Moreover, for each partial temporal behavior $pb \in \mathcal{PB}$ defined over $H' < H$, there exists a unique temporal evolution ρ_{pb} of $\mathcal{PL}$ such that ρ_{pb} represents the partial temporal behavior pb over the same horizon H'. That is, ρ_{pb} of $\mathcal{PL}$ represents the same valued intervals sequence in $\mathcal{P}$ limited to H' and the duration of ρ_{pb} is exactly the horizon H'. As a consequence, the winning strategy generated by UPPAAL-TIGA as a side effect of the verification process represents a flexible plan controller that achieves the planning goals maintaining the dynamic controllability during the overall plan execution. More formally, a plan controller $\mathcal{C}_f$ derived from a winning strategy f can be defined as follows.

Definition 1 *Given the reachability game $RG(\mathcal{PL}$,* Init*,* Safe*,* Goal*) defined as in Section 5, and the associated winning strategy f generated by UPPAAL-TIGA, a plan controller $\mathcal{C}_f$ is defined as follows: for each partial behavior $pb \in \mathcal{PB}$ over H', $\mathcal{C}_f(pb, H') = f(\rho_{pb}) = a_c$, where each controllable action a_c represents the associated values in $\mathcal{V}_1 \cup ... \cup \mathcal{V}_n$ (plus the special action λ corresponding to "just wait and do nothing"), $\mathcal{C}_f(pb, H')$ is undefined otherwise.*

As a consequence, the following theorem holds:

Theorem 1 *A controller $\mathcal{C}_f$ defined according to Definition 1 satisfies the following: (i) it correctly executes the plan $\mathcal{P}$ reaching the given planning goals and (ii) maintains the dynamic controllability property during plan execution.*

Moreover, it is also possible to define optimized controllers for flexible plans. Given a fixed temporal interval $[u, g]$ and a reachability game, UPPAAL-TIGA is able to generate a winning strategy f^* within that interval which minimize the plan execution duration [4]. Since a flexible plan is associated with a planning horizon $[0, H]$, an optimized controller can be generated with $[u, g] = [0, H]$. This allows to conclude the following:

Theorem 2 *Given a reachability game $RG(\mathcal{PL},$ Init, Safe, Goal$)$ defined as above within the temporal interval [u,g] = [0,H], the winning strategy f^* provided by UPPAAL-TIGA is time optimal and, because of Theorem 1, the derived controller $\mathcal{C}_{f^*}$ is also time optimal.*

6. EMPIRICAL RESULTS

This section investigates the assessment of the practical feasibility of the approach as well as the viability of the new KEEN functionality by using our robotic case study as a benchmark. Our aim is to test the controller generation performance in a real world scenario as well as to check whether the KEEN on-line control synthesis functionality is viable and compatible with the short latencies of a planning and execution cycle. Then, in this context, we want also to assess the controller synthesis overhead w.r.t. to the planning and verification costs. For this purpose, we introduce different planning/execution scenarios obtained by varying the problem complexity along the following dimensions: *plan length* by playing on both the number of pictures to be taken and the plan horizon; *plan flexibility* by modifying the allowed temporal tolerance for uncontrollable actions; *plan choices* by changing the number of communication opportunities. More specifically:

(1) *Plan Length.* We considered problem instances with an increasing number of requested pictures (from 1 to 5). At the same time, we consider flexible plans with a horizon length ranging from 150 to 550 seconds.

(2) *Plan Flexibility.* For each uncontrollable activity (i.e., GoingTo, MovingTo, TakingPicture, and Communicating), we set a minimal duration, but allow temporal flexibility on the activity termination, namely, the end of each activity has a tolerance ranging from 0 to 20 seconds. This temporal interval represents the degree of temporal flexibility/uncertainty that we introduce in the system.

(3) *Plan Choices.* We define from 1 to 4 visibility windows that can be exploited to communicate picture content.

Notice that an increasing number of communication opportunities raises the complexity of the planning problem with a combinatorial effect. More in general, among all the generated problem instances, the ones with higher number of required pictures, higher temporal flexibility, and higher number of visibility windows result as the hardest ones. In these scenarios, we analyzed the performance of our method considering model generation, controller synthesis, and plan execution. We used OMPS [14] as a CBTP Domain Independent Planner. The experiments have been ran on a MacBook Pro endowed with a Intel Core i5 (2.5GHz) processor and 4GB RAM. In what follows the reported timings are in seconds.

		1 wind	2 wind	3 wind	4 wind
#pic 1 H 150	bytes nr. of states	8108 29	8108 29	8671 33	8960 35
#pic 2 H 250	bytes nr. of states	10094 39	10370 39	10674 43	10936 45
#pic 3 H 350	bytes nr. of states	13051 53	13326 53	13603 57	13892 59
#pic 4 H 450	bytes nr. of states	14102 59	14378 63	14655 65	14943 69
#pic 5 H 550	bytes nr. of states	18151 69	16402 70	16678 73	16967 75

Figure 4. Size of generated models.

Model Generation. As a preliminary step of our evaluation, we considered the cost of generating the UPPAAL-TIGA model associated with the planning task. The dimension of the generated model is given in terms of number of generated states and the file size in bytes. As we can see in Fig. 4, for all the configurations, the generation process is very fast, taking less than 200ms for each instance, while the dimension of the generated model gradually grows with respect to the dimension of the flexible plan in terms of both plan length and number of visibility windows. The temporal flexibility does not affect the dimension of the generated models. Thus, we can conclude that model generation is not a critical step in our method.

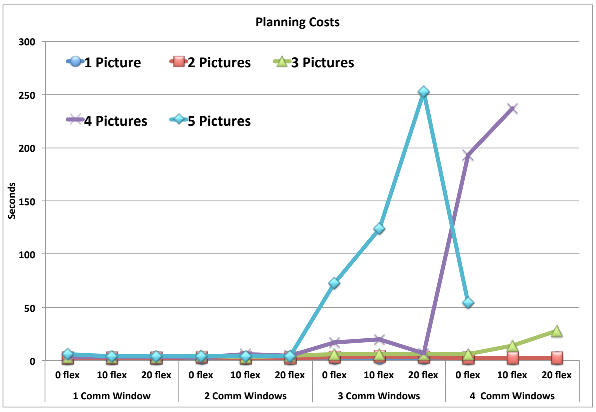

Figure 5. Plan generation cost varying the number of pictures, visibility windows and temporal flexibility.

Controller Synthesis. The cost of controller synthesis has been analyzed with respect to the cost of planning and the cost of plan verification (i.e., dynamic controllability check). The planning costs are collected in Fig. 5 where we can observe how the planner performance decreases with increasing communication windows and temporal flexibility. In particular, while simple instances (i.e., 1 or 2 communication windows) are solved in few seconds, the hardest ones require an additional planning effort (i.e., 4 communication windows and more than 3 required pictures). Actually, in this case some of the instances are not solved due to memory limit (in Fig. 5 values are missing for the last problem instances with 4 and 5 requested pictures).

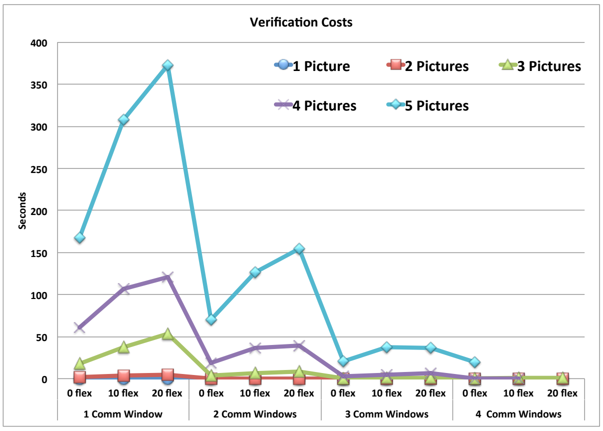

Figure 6. Dynamic Controllability verification cost.

The results collected for dynamic controllability checking (see Fig. 6) and strategies generation (see Fig. 7) show a quite different behavior. Interestingly, for hard problem instances flexible plan verification and strategy generation are very fast (3 and 4 communication windows in Fig. 6 and 7). While, with simpler instances (1 and 2 communication windows in Fig. 6 and 7), we do not observe the expected improvement in performance.

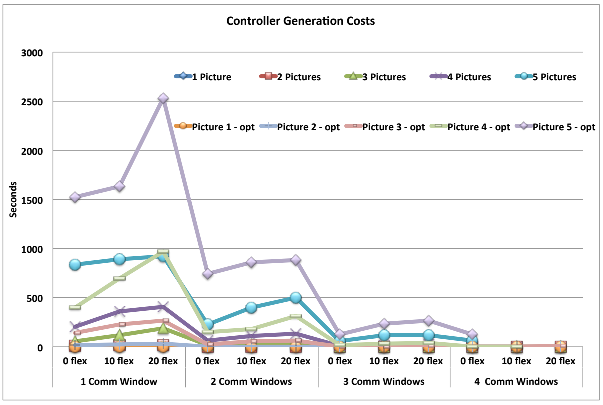

Figure 7. Strategies generation cost for both optimized and non optimized cases.

This is mainly due to the fact that simple planning problem instances are associated with few constraints to be considered, hence our planner can generate highly flexible temporal plans. However, this flexibility provides a wide search space to the verification tool reducing its performance. In contrast, harder planning problems lead the planner to produce flexible plans that are strongly constrained, i.e., with a lower degree of flexibility. This simplifies the UPPAAL-TIGA task which can check and generate strategies more quickly (see again the values for 4 communication windows in Fig. 6 and 7). Indeed, this is an expected behavior of the verification tool. In fact, the more non-determinism, the harder it is for UPPAAL-TIGA to generate strategies.

Plan Execution. As a final evaluation of the generated plan controllers, we considered the time needed for plan execution comparing optimized and non-optimized controllers. Besides providing empirical indications of the controllers effectiveness, our aim here is also to assess whether the gain in the execution performance in the optimized case can justify the generation cost overhead. The execution has been simulated in UPPAAL-TIGA considering time average and variance of 20 runs and randomly generating the temporal occurrences (within their duration intervals) of the uncontrollable events mentioned in the plan. In Fig. 8, the collected results show a slight enhancement in time efficiency in the optimized version, but this gain seems negligible, in particular when compared with the generation cost. This seems to suggest that sub-optimal controllers provide a better trade-off between control synthesis and plan execution.

Discussion. The experimental results show the practical feasibility of the TGA approach in increasingly complex instances of a real-world robotic case study. The collected data show also an interesting relationship among the complexities of the planning tasks, generated flexible plans, and generated controllers. We observe that additional efforts during the planning phase usually reduces the cost of plan verification and control synthesis and, vice-versa, simpler planning problems are usually associated with more complex controller synthesis tasks. Furthermore, more complex planning tasks should be associated with longer execution latencies, hence if we contrast the controller generation time w.r.t. the planning horizon length (assuming a comparable time available for planning), taking apart the hardest instances (5 pictures, and 3 or 4 pictures with less than 3 visibility windows), all the other cases are treatable. Moreover, if we consider plan complexity (visibility windows) and control synthesis cost overhead with respect to the plan generation cost, we observe that with few required pictures (e.g., 1 or 2 pictures), the control synthesis cost remains acceptable for all the visibility windows, while, with additional visibility windows (e.g., 4 visibility windows), additional pictures can be introduced. In conclusion, this empirical analysis shows how the control synthesis overhead remains very low in most of the considered instances. In particular, the controller synthesis method is compatible with the performance required by a fast short-horizon planner.

7. CONCLUSION

The paper has presented the exploitation of a V&V technique to enrich the KEEN environment enabling the automatic synthesis of controllers for flexible temporal plans. While flexible temporal plan execution is usually addressed using temporal constraint networks methods and algorithms to reduce the plan in a dispatchable form, this new functionality proposes an alternative and novel technique based on the generation of a winning strategy with TGAs. According to this approach, the plan execution problem can be solved completely as a side effect of dynamic controllability checking; hence, all the plan execution decisions can be available before the plan execu-

1 Comm. Window				2 Comm. Windows				3 Comm. Windows				4 Comm. Windows			
pic	0s flex	10s flex	20s flex	pic	0s flex	10s flex	20s flex	pic	0s flex	10s flex	20s flex	pic	0s flex	10s flex	20s flex
1	139±0	146±3	148±1	1	131±0	142±7	141±3	1	132±0	137±6	145±3	1	116±0	139±7	138±8
2	243±0	211±6	243±6	2	198±0	232±13	238±11	2	213±0	231±8	230±8	2	157±0	177±11	184±12
3	242±0	291±2	339±7	3	238±0	313±5	336±9	3	231±0	284±6	337±12	3	230±0	211±9	224±11
4	431±0	427±5	542±7	4	421±0	415±10	437±8	4	423±0	401±6	423±6	4	409±0	403±6	N/A
5	535±0	537±9	542±7	5	507±0	527±8	536±12	5	538±0	525±7	528±10	5	529±0	N/A	N/A
Optimal				Optimal				Optimal				Optimal			
1	98±0	118±7	132±4	1	81±0	97±6	121±8	1	78±0	87±4	108±3	1	66±0	94±6	99±12
2	173±0	194±11	229±16	2	167±0	211±9	218±13	2	145±0	176±17	201±7	2	142±0	153±11	167±9
3	237±0	286±9	332±12	3	231±0	307±8	327±4	3	227±0	279±4	331±14	3	223±0	209±4	218±15
4	428±0	428±6	432±7	4	418±0	411±11	430±12	4	420±0	397±12	421±7	4	404±0	401±8	N/A
5	512±0	527±14	531±9	5	494±0	518±6	521±10	5	528±0	511±8	507±9	5	511±0	N/A	N/A

(a)

(b)

Figure 8. Plan controllers execution performance (average durations and variances).

tion with an acceptable overhead with respect to the planning activity. It is worth mentioning how the KEEN system and the discussed method rely foremost on off-the-shelf planning/verification tools such as the OMPS and UPPAAL-TIGA tool-chain and on an encoding tool tailored to translate the plan specification into TGAs.

As an ongoing work we are making the whole environment more robust and self contained. We plan also to realize in the near future additional functionalities to facilitate the use of the environment to users of different skills. Additionally we plan to enhance knowledge engineering support to domain modeling functionalities.

Acknowledgment. Amedeo Cesta is partially supported by MIUR under the PRIN project 20089M932N (funds 2008). Alberto Finzi is partially supported by EU under the AIRobots project (Contract FP7.248669). Andrea Orlandini has been supported by a grant within "Accordo di Programma Quadro CNR-Regione Lombardia: Progetto 3".

REFERENCES

1. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, pages 121–125. Springer, 2007.
2. S. Bensalem, L. de Silva, M. Gallien, F. Ingrand, and R. Yan. "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In *i-SAIRAS-10. Proc. of the 10th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2010.
3. G. Bernardi, A. Cesta, and G. Cortellessa. Deploying RAXEM2: Planning Improvements in Daily Work Practice. In *SPARK-09. Scheduling and Planning Applications woRKshop at ICAPS*, Thessaloniki, Greece, 2009.
4. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, pages 66–80. Springer-Verlag, 2005.
5. A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael. A Goal-Oriented Autonomous Controller for Space Exploration. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
6. A. Cesta, G. Cortellessa, M. Denis, A. Donati, S. Fratini, A. Oddi, N. Policella, E. Rabenau, and J. Schulster. MEXAR2: AI Solves Mission Planner Problems. *IEEE Intelligent Systems*, 22(4):12–19, 2007.
7. A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. MRSPOCK: Steps in Developing an End-to-End Space Application. *Computational Intelligence*, 27(1), 2011.
8. A. Cesta, A. Finzi, S. Fratini, and A. Orlandini. Enriching apsi with validation capabilities: the keen environment and its use in robotics. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
9. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press, 2010.
10. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review*, 25(3):299–318, 2010.
11. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107:111–137, 2011.
12. A. Cesta and S. Fratini. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *PlanSIG-08. Proc. of the 27th Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12*, 2008.
13. S. Fratini, A. Cesta, R. De Benidictis, A. Orlandini, and R. Rasconi. APSI-based deliberation in Goal Oriented Autonomous Controllers. In *ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
14. S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.
15. E. Gat. On Three-Layer Architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press, 1997.
16. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

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

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

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

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

21. A. Orlandini, A. Finzi, A. Cesta, and S. Fratini. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, pages 233–245. Springer, 2011.

22. F. Py, K. Rajan, and C. McGann. A Systematic Agent Framework for Situated Autonomous Systems. In *AAMAS-10. Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

23. J. Shah and B. C. Williams. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *ICAPS-08*, pages 322–329, 2008.

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