ENRICHING APSI WITH VALIDATION CAPABILITIES: THE KEEN ENVIRONMENT AND ITS USE IN ROBOTICS

Andrea Orlandini[1,3], Alberto Finzi[2], Amedeo Cesta[1], Simone Fratini[1], and Enrico Tronci[4]

[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*
[3]*CNR – Consiglio Nazionale delle Ricerche, ITIA, Milan, Italy – andrea.orlandini@itia.cnr.it*
[4]*DI – Università di Roma "La Sapienza", Rome, Italy – enrico.tronci@di.uniroma1.it*

ABSTRACT

This paper presents the KnowledgE ENgineering (KEEN) design support system in which Validation and Verification (V&V) methods are used to strengthen on-ground development of software for plan-based autonomy. In particular, the paper describes a collection of verification methods, based on Timed Game Automata (TGA), deployed for the design and development of timeline-based Planning and Scheduling (P&S) applications within the APSI-TRF framework. The KEENs V&V functionalities are illustrated describing software development to synthesize plans for a planetary rover.

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

1. INTRODUCTION

Mission operations support tools using robust Planning and Scheduling (P&S) applications have achieved increasing success during the last decades [24, 23, 17, 13, 5, 6, 12]. It is worth noting how all of them use a domain modeling and reasoning engine based on the concept of timelines (see for example [15]). Being space missions very demanding in terms of costs, the validation of the outcome of any involved sub-activity represents a key issue. In particular, the issue of Verification and Validation (V&V) is of great importance to enhance the reliability and maintainability of supporting tools [21]. Particularly relevant is the possibility of having tools that manipulate the planning results demonstrate V&V properties by using orthogonal technology.

The synthesis of knowledge engineering environments in which constraint programming and V&V techniques may concur in creating enhanced software development environment for P&S is discussed in [9]. The present work describes a prototype of a software environment along that idea. The KnowledgE ENgineering (KEEN) design support system is described in which V&V methods are used to support the development of P&S applications. The KEEN system is built around the APSI-Timeline Representation Framework (APSI-TRF) [11]: a timeline-based support to model domain that is coupled with a planner to synthesize solution for a planning problem. [1]

The KEEN system is composed of different V&V modules implementing design support functionalities. In particular, a TGA Encoding module is deputed to implement a translation from P&S specification to TGA. The encoding method is the same presented in [7], sharing the same formal results presented in [8, 10]. Then, relying on that encoding and results, the KEEN system is endowed with: a Domain Validation module, to support the model building activity providing a tool to assess models quality; a Planner Validation module, to assess the P&S solver with respect to system requirements; a Plan Execution Validation module, to check whether the proposed solution plans are suitable for actual execution or not. In order to implement the modules functionalities, the verification tasks are performed by means of UPPAAL-TIGA [1], a toolbox for the specification, simulation, and verification of real-time games. In this paper, we show how various modules can be obtained by exploiting verification tasks performed by means of UPPAAL-TIGA and the details of its use to support a planetary rover mission operations. The KEEN system represents a general tool to support timeline-based planning.

The paper is organized as follows: we first introduce the robotic domain and the associated planning problem sketched as a timeline-based planning and execution problem. Then, we describe our verification method for flexible timeline-based plans presenting the KEEN design support system. The KEEN's V&V functionalities are then illustrated describing the design and development of a planetary rover controller endowed with a P&S system. Some conclusions end the paper.

2. PLAN BASED ROBOT CONTROL

This work takes as reference a plan-based autonomous system organized in a generic three layered control architecture [16] that combines a physical layer, a functional layer, and a deliberative layer. The functional layer provides an abstraction of the physical system wrapping the controllers for the robotic devices (e.g., PTU, Camera, navigation, etc.). A generic deliberative layer is composed by a planning and scheduling module and a executive system. The generic connection among the modules can be sketched as in Figure 1. The planning and scheduling module is responsible for mission and task planning:

[1]The Advanced Planning and Scheduling Initiative (APSI) has been an ESA project from 2006 to 2009. The Timeline Representation Framework has been developed within APSI by the CNR-ISTC group. The KEEN environment is developed outside ESA programs.

given a set of mission goals it generates temporal plans of actions to be delivered to the executive system. The executive system is responsible for plan monitoring, command dispatching and fault detection.

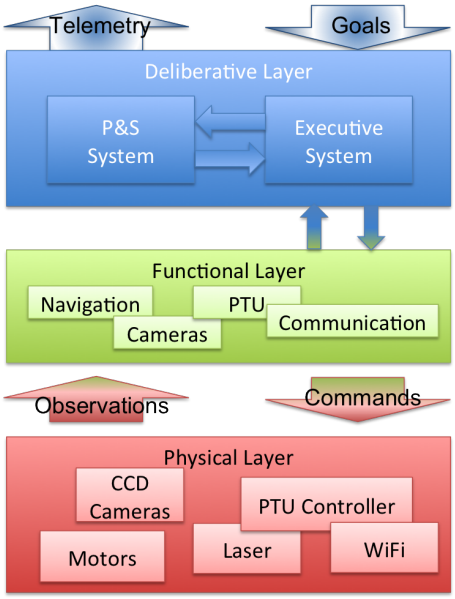

Figure 1. A control architecture for a robotic platform

An Example of Robotic Domain. In the paper we use a real running example taken from the current GOAC project [4] integrated demo.[2] The experimental setting includes 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. A safe PTU position is assumed to be with *pan* and *tilt* values as $(0, 0)$. 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 configurations. A possible mission actions 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 taken picture 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 in order to maintain safe configurations and do not affect actions execution effectiveness. Namely, the following conditions must hold during the overall mission:

- **(C1):** While the robot is moving the PTU has to be in the safe position;
- **(C2):** The robotic platform can take a picture only if the robot is still in one of the requested location 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.

In real domains like this, it is not possible to determine the actual execution duration of each task in advance. Thus, termination commands of each robot task are to be considered as uncontrollable to the executive system. That is, since the termination time of each task is not fixed, task duration may vary within a temporal interval $[lb, ub]$ (lb is the minimum duration while ub is the maximum).

3. TIMELINE-BASED PLANNING AND EXECUTION

Timeline-based planning is an approach to temporal planning which has been applied to the solution of several space planning problems – e.g., [23, 17, 25, 13, 6, 12]. This approach pursues the general idea that P&S for controlling complex physical systems consist in the synthesis of a set of desired temporal behaviors for system features that vary over time.

State variables and timelines. The features to be controlled in a domain can be modeled as a set of temporal functions whose values over a time horizon have to be planned for. Such functions are synthesized during problem solving by posting planning decisions that specify the values taken by temporal functions over time intervals. The evolution of a single temporal feature over a time horizon is called the *timeline* of that feature, i.e., a scheduled set of planning decisions.

The time varying features are called multi-valued *state variables* as in [23]. As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal occurrences of planned decisions. Such causal laws are specified for the state variables in a *domain specification* which describes the operational constraints in a given domain. In this context, the task of a planner is to find a sequence of decisions that brings timelines into a final desired set always satisfying the domain specification.

We assume that the temporal features we want to represent as state variables have a finite set of possible values assumed over temporal intervals. The temporal evolutions are sequences of operational states – i.e., stepwise constant functions of time. Operational constraints specify which value transitions are allowed, the duration of each valued interval (i.e., how long a given operational status can be maintained) and synchronization constraints between different state variables.

More formally, a state variable is defined by a tuple $\langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$ where: (a) $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *values*; (b) $\mathcal{T} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$ is the *value transition* function;

[2] Thanks to Felix Ingrand and Lavindra De Silva from LAAS-CNRS for the time spent to explain us the details of their robotic platform.

(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).

Timeline specification for the robotic domain. To obtain a timeline-based specification of our robotic domain, we consider a set of state variables to represent time varying features for the temporal occurrence of navigation, PTU, camera, and communication operations as well as communication opportunities. In this regard, we consider five different state variables, i.e., *RobotBase*, *Platine*, *Camera*, *Communication* and *Orbiter Visibility*. In Figure 2, we detail the values that can be assumed by these state variables, their durations and the allowed value transitions in accordance with the mission requirements and the robotic physics. Notice that dotted arrows represent uncontrollable transitions.

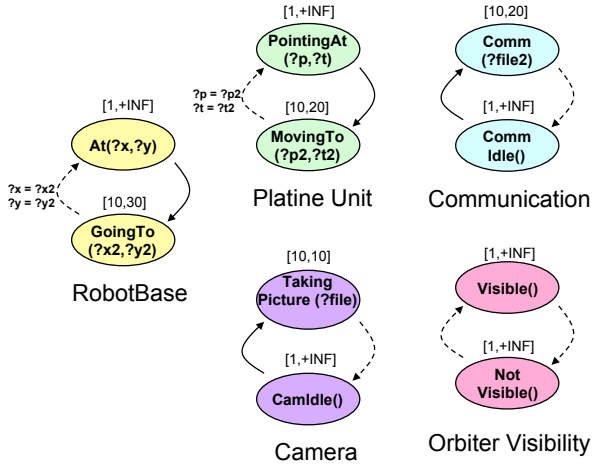

Figure 2. Value transitions for state variables describing the robotic platform activities and the orbiter visibility events (Temporal durations are stated in seconds)

The robotic platform can be in a certain position (*At(x,y)*) or moving to a certain 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)* value. The camera can take a picture of a given object in a position $\langle x, y \rangle$ with the PTU requested in $\langle pan, tilt \rangle$ and store it as a file in the onboard memory (*TakingPicture(file-id, x, y, pan, tilt)*) or be in an idle state (*CamIdle()*). Similarly, the communication facility can be operative and dumping a given file (*Communicating(file-id)*) or be in an idle status (*CommIdle()*). Finally, the *Orbiter Visibility* state variable maintains the visibility of the orbiter and its allowed values, i.e., *Visible* and *Not-Visible*, representing external constraints for the planning problem. In particular, these values represent contingent communication opportunities for the rover.

Synchronizations for the robotic domain. In timeline-based planning, operational constraints in the domain are described by means of *Synchronizations*. Indeed, synchronizations specifies temporal and causal constraints among values taken over different timelines (i.e., patterns of legal occurrences of operational states across different timelines).

The Figure 3 exemplifies the use of synchronizations in our case study domain. The following synchronizations are represented in the figure: the *PointingAt(0,0)*[3] value must occur during a *GoingTo(x,y)* value (**C1**); the *At(x,y)* and *PointingAt(pan,tilt)* values must occur during a *TakingPicture(pic,x,y,pan,tilt)* value (**C2**); the *Communicating(pic)* must occur after a *TakingPicture(pic,x,y,pan,tilt)* (**C3**); the *At(x,y)* value must occur during a *Communicating(file)* (**C4**); the *Visible* value must occur during a *Communicating(file)* (**C5**). (C1) and (C4) represent safety conditions: when moving or communicating the rover must be in a safe configuration (PTU unit in (0,0) when moving or not moving when communicating). (C2) and (C5) represent temporal synchronizations among different activities (to take a picture the rover must be in the proper place and configuration in the right time (C2) and dumps must occur when the orbiter is visible (C5)). (C3) describes a pure cause-effect relationships between two activities: pictures must be dumped once stored.

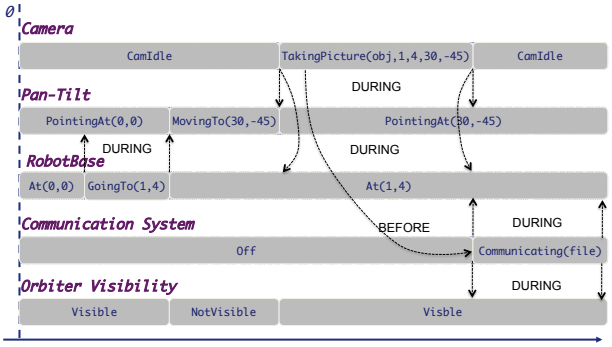

Figure 3. An example of timeline-based plan with synchronizations in our case study domain.

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 Figure 2).

Timeline-based planning. In timeline-based planning, *goals* are expressed as desiderata of timeline values in temporal intervals and the task of the planner is to build a set of timelines describing valid sequences of values that achieve such 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, as it happens in case of a least commitment solving approach for instance, we refer to timelines as *time flexible* and to plans resulting from a set of flexible timelines as *flexible plans*.

The process of *solution extraction* from a plan is the process of computing (if exists) a *valid* and completely specified set of timelines from a given set of time-flexible timelines. A solution is *valid* with respect to the associated domain theory if every synchronization is satisfied.

[3] *(pan,tilt) = (0,0)* is assumed as the reference/safe position for the PTU

Plan execution. During plan execution, the plan is under the responsibility to the executive system (recall Figure 1) that forces value transitions over timeline dispatching commands to the functional layers while continuously accepting observations and, thus, monitoring the plan execution. A well known problem with execution is that not all the value transitions are under responsibility of the executive but event exists that are under control of *nature*. As a consequence, an executive cannot completely predict the behavior of the controlled physical system because the duration of certain processes or the timing of exogenous events is outside of its control. In such cases, the values for the state variables that are under the executive scope should be chosen so that they do not constrain uncontrollable events. This is the *controllability problem* defined, for example, in [26] where *contingent* and *executable* processes are distinguished. The contingent processes are not controllable, hence with uncertain durations, while the executable processes are fully under the control of the executive system. Controllability issues underlying a plan representation have been formalized and investigated for the Simple Temporal Networks with Uncertainty (STNU) representation in [26] where basic formal notions are given for *dynamic* controllability (see also [22]). In [7], the controllability problem has been extended for the timeline-based framework. The dynamic controllability is a quite important property that guarantees robust execution against temporal uncertainty and not fully controllable systems.

4. TIMED GAME AUTOMATA AND REACHABILITY GAME

A TGA [20] is an automaton equipped with clocks that grow continuously in time while the automaton is in any of its locations. The values of the clocks may interfere with the transitions by appearing in *guards*, which are the enabling conditions of the transitions. Thus, a transition may take place, for example, only if some clock value has passed a certain threshold. Transitions may as well reset clocks. Moreover, state *invariants* may be defined constraining the temporal behaviors. Finally, a set of actions is considered to label transitions. Actions can be either *controllable* or *uncontrollable*. This defines a two player games with on one side the *controller* (mastering the controllable transitions) and on the other side the *environment* (mastering the uncontrollable transitions). A TGA state is identified by the location of the TGA and the *valuation* of the clocks (i.e., a value assignment for each clock). An admissible state is a state respecting all the invariants. A TGA can either let time progress or do a discrete transition and reach a new location. A *run* of a TGA is a finite or infinite sequence of alternating time and discrete transitions. A *network* of TGA (nTGA) is a finite set of TGA evolving in parallel with a CCS style semantics for parallelism. Namely, at any time, only one TGA in the network can change location, unless a synchronization on labels takes place. In the latter case, the two automata synchronizing on the same label move together. Note that time does not elapse during synchronizations.

Given a TGA $\mathcal{A}$ and three symbolic state configurations *Init*, *Safe*, and *Goal*, the *reachability control problem* [3] or reachability game $RG(\mathcal{A}, Init, Safe, Goal)$ consists in finding a *strategy* f such that $\mathcal{A}$ starting from *Init* and supervised by f generates a winning run that stays in *Safe* and enforces *Goal*. A strategy is a partial mapping f from the set of runs of $\mathcal{A}$ starting from *Init* to the set of controllable actions plus the special action λ (a special symbol that denotes "do nothing and just wait"). For each state s in a finite run of $\mathcal{A}$, the strategy f may say (i) no way to win ($f(s)$ is undefined), (ii) do nothing, just wait ($f(s) = \lambda$), or (iii) execute the discrete, controllable transition labeled by l ($f(s) = l$). The restricted behavior of a TGA $\mathcal{A}$ controlled with some strategy f is defined by the notion of *outcome*. The outcome of a strategy f in $\mathcal{A}$ is defined as the subset of all the possible runs of $\mathcal{A}$ that can be generated executing the uncontrollable actions or the controllable actions provided by the strategy f. A strategy f is a *winning strategy* if all runs in the outcome of f reach the goal configuration while enforcing the safe configuration.

5. THE KEEN ENVIRONMENT AND ITS USE

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 [9]. 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. In particular we are envisaging the use in a robotic mission as the one sketched in section 2. After quite an amount of work in developing specific applications for ESA (e.g., [5, 2, 6]) and in creating the APSI-TRF infrastructure for P&S [11] 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-creating within the APSI-TRF structured style of implementation a general purpose timeline planner like OMPS [15]. The use of the APSI-TRF is shown in the central block of Figure 4: 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 (the 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 [14] but we can easily taylor such block for different ground segment needs. Around this core tool we have built additional engineering services based on our recent work on V&V issues.

In what follows, we present a prototype of the KnowledgE ENgineering (KEEN) design support system which is composed of different V&V modules implementing different design support functionalities (see Figure 4).

A *TGA Encoding* module is deputed to implement a translation from P&S specification to TGA. The encoding method is the same presented in [7] allowing to share the same formal results presented in [8, 10]. The other

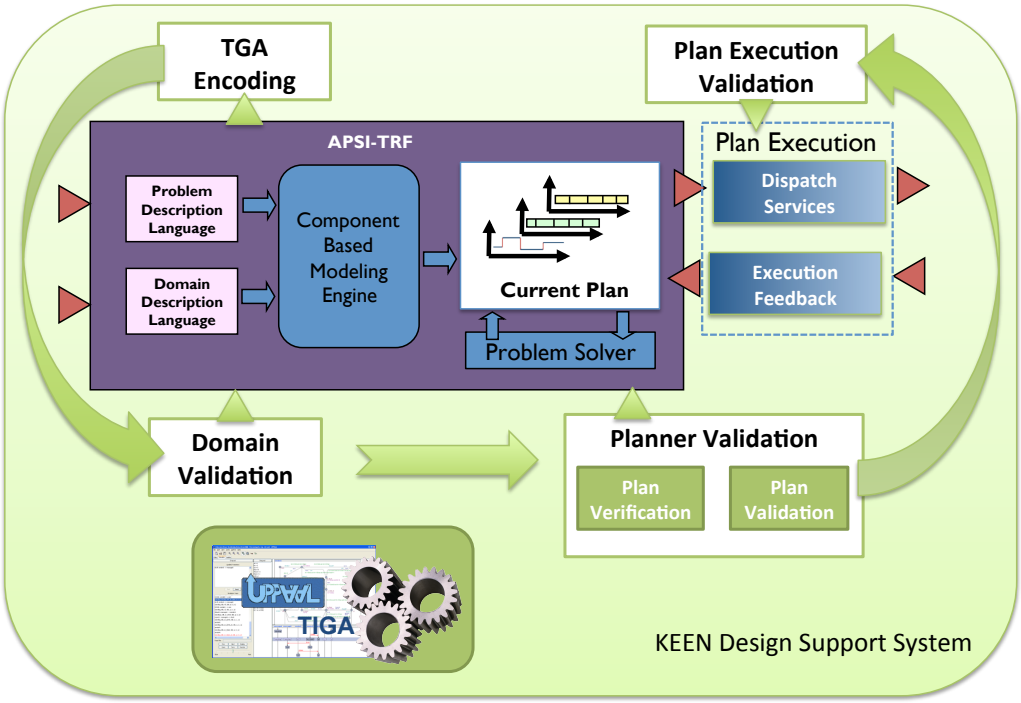

Figure 4. The KnowledgE ENgineering (KEEN) Design Support System.

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 [19] 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. In the following, each module is described providing also a formal account of the approach.

TGA Encoding. As discussed in Section 4, TGA are particularly suitable for modeling real-time systems and controllability problems, considering uncontrollable activities as *adversary moves* within a game between the controller and the environment. The KEEN TGA encoding module translates flexible timeline-based plans, state variables and domain theory descriptions into set of TGA (nTGA) as in [7]. To this end, the following translations are needed.

First, 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 guards and invariants are defined according to lower and upper bounds of flexible timed intervals. In Figure 5, the UPPAAL-TIGA model of a possible flexible plan for the robotic case study is depicted. Notice that straight and dotted arrows have been used to represent, respectively, controllable and uncontrollable transitions.

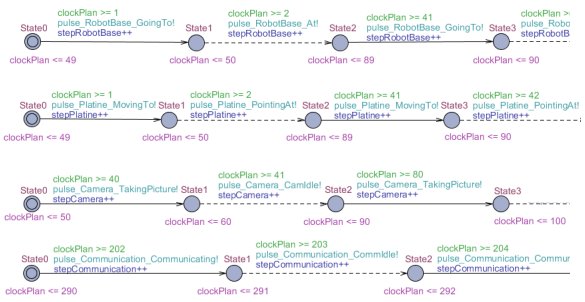

Figure 5. A TGA model representing a flexible plan for the robotic domain

Then, the related set of state variables *SV* is mapped into a nTGA *StateVar*. Basically, a one-to-one mapping from state variables descriptions to TGA is performed. As an example, the UPPAAL-TIGA specification for the *RobotBase* state variable is given in Figure 6. In this encoding, the flexible plan *view* of the world is represented by partitioning possible value transitions into controllable and uncontrollable ones. This is necessary to model the actual context in which the plan should be executed. For instance, the *Orbiter Visibility* state variable is considered as completely uncontrollable. Finally, an *Observer* automaton is introduced in order to check for value constraints violations as well as synchronizations violations. In particular, two locations are considered: an Error location, to state that constraint/synchronization has been violated and a Nominal (OK) location, to state that the plan behavior is correct. In Figure 7, an excerpt of the TGA monitor specification is provided. In this kind of approach, as usual, the *Observer* is defined as fully uncontrollable.

The nTGA $\mathcal{PL}$ composed by the set of automata *StateVar*

```
process RobotBase() {

  state GoingTo {clockRobotBase <= 30}, At;

  init At;

trans
  GoingTo -u-> At {guard clockRobotBase >= 10;
      sync pulse_RobotBase_At?; assign clockRobotBase := 0,
                  RobotBaseGoingTo := false, RobotBaseAt := true; },

  At -> GoingTo {guard clockRobotBase >= 1;
     sync pulse_RobotBase_GoingTo?; assign clockRobotBase := 0,
                  RobotBaseGoingTo := true, RobotBaseAt := false; };
}
```

Figure 6. TGA specification for the "RobotBase" state variable

```
process monitor() {  state OK,ERR;
  init OK;

  trans
    OK -u-> ERR {guard (stepRobotBase == 0) and not (RobotBaseAt);},
    OK -u-> ERR {guard (stepRobotBase == 1) and not (RobotBaseGoingTo);},
    OK -u-> ERR {guard (stepRobotBase == 2) and not (RobotBaseAt);},
    ...
    // -- DT --
    OK -u-> ERR {guard (clockRobotBase > 0) and (clockPlatine > 0)
                  and  (RobotBaseGoingTo) and not (PlatinePointingAt);},
    OK -u-> ERR {guard (clockCamera > 0) and (clockRobotBase > 0)
                  and  (CameraTakingPicture) and not (RobotBaseAt);  },
    ...
    ERR -u-> ERR { };
}
```

Figure 7. The "Observer" TGA specification

$\cup$ *Plan* $\cup \{\mathcal{A}_{Obs}\}$ models flexible plan, state variables and domain theory descriptions. In [8], the following has been demonstrated:

Theorem 1 *The nTGA $\mathcal{PL}$ describes all and only the behaviors implemented by the flexible plan $\mathcal{P}$.*

Thus, the KEEN TGA Encoding module provides a proper description of P&S specifications in terms of TGA. Such encoding is exploited by the other KEEN modules to implement their functionalities.

Domain Validation. In a similar way to [18], the TGA encoding can be exploited in order to validate planning domains with respect to, e.g., undesired behaviors, safety properties, etc. In fact, domain validation is the process of assessing domain models. In this regard, the KEEN Domain Validation module is to support knowledge engineers in the process of eliciting, refining and correcting the domain model w.r.t. safety- and system-critical requirements. To implement such a functionality, the nTGA *Dom* = *StateVar* $\cup \{\mathcal{A}_{Obs}\}$ can be considered. In fact, the following can be obtained as a result of Theorem 1:

Proposition 1 *The set of TGA* Dom = StateVar $\cup \{\mathcal{A}_{Obs}\}$ *represents all and only the possible behaviors described by the associated planning domain.*

Thus, given a system property F, suitably represented by a CTL formula ϕ, verifying ϕ in *Dom* corresponds to validate the planning domain with respect to the same property F. That is, it is possible to check the correctness of the domain asking UPPAAL-TIGA to verify the formal property ϕ over the nTGA *Dom*.

Among the most relevant properties to be checked, the violation of mutual exclusion of timeline's allowed values is very important. In fact, such test is useful for detecting an incomplete specification of synchronizations in the planning domain theory. For instance, if successfully checked, the property ($E\Diamond$ *RobotBase.GoingTo and Communication.Communicating)*, which reads *there exists a trace where at some point in time the rover is moving while communicating*, represents a flaw in the domain. That is, the (**C4**) domain constraints might be violated. Of course, the planning model defined in Section 3 does not satisfy such a property (i.e., the domain is correct) but, in general, if this kind of test succeeds, then knowledge engineers might have defined wrong or incomplete synchronizations in the planning domain.

The reachability test is another important check. That is, the reachability of each allowed value in the domain is checked starting from one specific initial state or from each possible initial state. As an example, the *Communication* value should be reachable after a *TakingPicture*. This corresponds to check the following formula: $A\Box$ *Camera.TakingPicture and E*$\Diamond$ *Communication.Communicating*. Performing this test allows to verify whether the model guarantees that is always possible to communicate after a picture has been taken. In general, finding that a certain value is unreachable may suggest that inconsistent specifications are contained in the planning domain.

Planner validation. In order to validate the planner, we are interested in checking that the planning engine works properly. In this sense, the planner design activity should be supported by providing effective methods to validate the solver and the generated solutions. One aspect of its correctness is the capability of generating a correct plan whenever there is one. In addition, also the quality of generated plans should be checked.

For this purpose, two important submodules are required: Plan Verification, which systematically analyzes the solutions proposed by the planner itself, and Plan Validation, which allows to assess the plan quality. Errors or negative features possibly found in the generated plans could help knowledge engineers to revise the model (back to the domain validation step), the heuristics, or the solver. Furthermore, plan V&V is also to analyze the produced plans with respect to execution controllability issue. As for the domain validation module, the KEEN Plan Verification and Plan Validation modules have been implemented through the same verification method presented above. In fact, 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*, the following has been demonstrated ([8]):

Theorem 2 *Given $RG(\mathcal{PL}$,* Init*,* Safe*,* Goal*) defined considering* Init*,* Safe *and* Goal *as above, winning the game implies plan validity for $\mathcal{P}$.*

Plan verification. To implement the Plan Verification module, the $RG(\mathcal{PL}$, *Init*, *Safe*, *Goal*) defined above can be solved by means of UPPAAL-TIGA. Then, to solve the reachability game, we ask UPPAAL-TIGA to check the CTL formula $\Phi =$ *A [Safe U Goal]* in $\mathcal{PL}$. In fact,

this formula states that along all the possible evolutions, $\mathcal{PL}$ remains in *Safe* states until *Goal* states are reached. Thus, if the solver verifies the above property, then the flexible temporal plan is valid. Whenever the flexible plan is not verified, UPPAAL-TIGA produces an execution strategy showing one temporal evolution that leads to a fault. Such a strategy can be analyzed in order to check for plan weaknesses or for the presence of flaws in the planning model.

In order to show the feasibility of the application of such a method to the robotic case study introduced in Section 2, we show the verification method performance presenting and discussing some experimental results collected by verifying the flexible plans generated in different execution contexts obtained varying both plan horizon and temporal flexibility settings. In particular, we consider the following settings:

- *Flexibility.* For each activity (i.e., GoingTo, MovingTo, TakingPicture, Communicating), we set a fixed minimal duration, but we allow temporal flexibility on the activity termination. More precisely, the end of each activity has a tolerance ranging from 10 up to 20 seconds. For instance, considering 15 seconds of temporal flexibility for the termination of the TakingPicture activity allows durations within [10,25] seconds. Such interval represents a temporal uncertainty introduced in the system.
- *Horizon.* We consider flexible plans with a horizon length ranging from 1 to 20 picture requested (i.e., from 2 minutes up to about 1 hour).

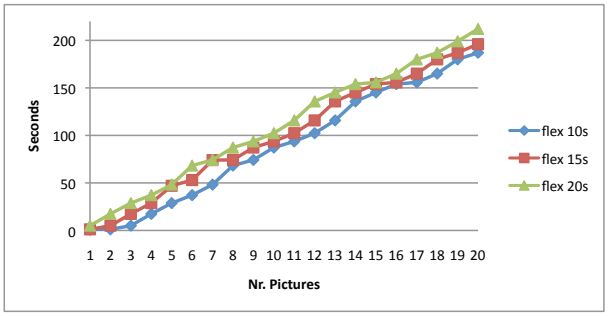

Figure 8. Verification tool performances varying temporal flexibility and planning horizon

As a result, verification time is linear in the size of the plan horizon, while increasing temporal flexibility does not affect the overall performance (see Figure 8).

Plan validation. It is worth reminding that while addressing real applications, besides synchronization constraints we may need also to take into account other constraints which cannot be naturally represented as temporal synchronizations among specific activities. Nevertheless, these constraints, that we call *relaxed constraints*, define a kind of desiderata on the global behavior of the generated plan. These requirements are not explicitly represented in the planning model as structural constraints, but rather treated as meta-level requirements to be enforced by the planner heuristics and optimization methods. Then, to implement the Plan Validation module, it is possible to apply the same verification process as in plan verification, verifying not only the plan correctness, but also other domain-dependent constraints, i.e., the relaxed constraints. In general, the additional properties to be checked carry a low additional overhead to the verification process. Thus, the verification tool performances are not affected.

In the robotic case study, we are able to successfully check that no unnecessary tasks have been planned. For instance, a communication task should be considered only if a picture is taken. Also, unnecessary robot navigation can be considered. In fact, the plan should not consider *GoingTo(x,y)* task unless that a picture is required for such a location. In general, the execution of such tasks may affect the overall mission performance. Thus, this kind of validation results to us as a really important step in assessing the plan quality, that is, the planner effectiveness.

Plan execution validation. In general, plan validation does not guarantee the robustness of plan execution, indeed, a valid plan can be brittle at execution time due to environment conditions that cannot be modeled in advance (e.g., *disturbances*). The KEEN system can also be deployed for plan execution validation.

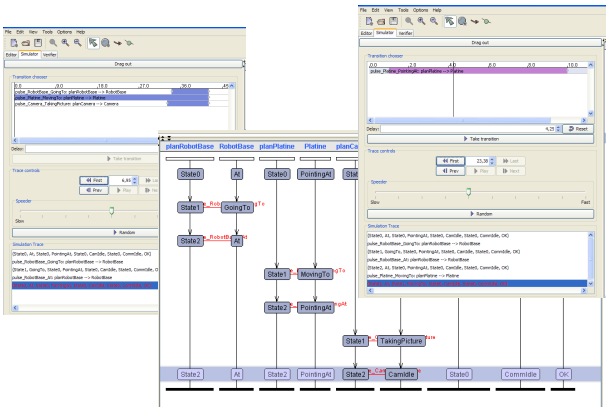

Figure 9. A plan execution simulated by UPPAAL-TIGA

In particular, the KEEN Plan Execution Validation module allows to simulate plan executions through UPPAAL-TIGA ingesting both controllable and uncontrollable events. Indeed, the UPPAAL-TIGA Simulator allows to simulate the TGA model based on concrete trace that a user can choose or that can be randomly generated. In this respect, the Simulator is able to support users to select transitions to fire and, then, *at what time* they will be fired. Figure 9 shows the UPPAAL-TIGA simulator graphical elements.

The UPPAAL-TIGA simulator is extremely useful at debug time to understand why a winning condition cannot be met, i.e., why a flexible plan fails in reaching a desired goal. Moreover, if a flexible plan cannot be successfully verified, UPPAAL-TIGA allows also to simulate the run of a dual strategy allowing to test intuitive strategies or to discover tactics used by the environment to defeat the controller. In other words, it is possible to get evidences of unforeseen behaviors or unexpected events. After a flexible plan has been verified by UPPAAL-TIGA through the verification method discussed above, the Simulator can also be used to show the dynamically

controllable strategy guaranteeing goals reachability and avoiding synchronization violations. That is, it is also possible to validate the execution strategy of the plan.

6. CONCLUSIONS

As described in several recent papers, e.g., [9], we have the goal of investigating the common ground between P&S and V&V. In particular the possible methodological integration is seen as beneficial to create a new generation of knowledge engineering environments.

This paper introduces our current software environment for supporting the development of plan-based robotic control software. In particular, we have shown how generic V&V tools and specific research results on the verification of timeline-based plans can be contextualized to create a larger environment to support on-ground planning and execution for a robotic mission.

Acknowledgment. Cesta, Fratini, and Tronci are partially supported by EU under the ULISSE project (Contract FP7.218815). Cesta and Fratini are also partially supported by MIUR under the PRIN project 20089M932N (funds 2008). Finzi is partially supported by EU under the AIRobots project (Contract FP7.248669). Orlandini is currently 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. 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.
3. 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.
4. 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.
5. 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.
6. A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. MR-SPOCK: Steps in Developing an End-to-End Space Application. *Computational Intelligence*, 27(1), 2011.
7. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible Timeline-Based Plan Verification. In *KI 2009*, volume 5803 of *LNAI*, 2009.
8. 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.
9. 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.
10. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107:111–137, 2011.
11. 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.
12. S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th International Conference on Automated Planning and Scheduling*, 2010.
13. J. Frank and A. Jonsson. Constraint Based Attribute and Interval Planning. *Journal of Constraints*, 8(4):339–364, 2003.
14. 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.
15. 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.
16. E. Gat. On Three-Layer Architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press, 1997.
17. A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proc. of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 177–186, 2000.
18. L. Khatib, N. Muscettola, and K. Havelund. Mapping Temporal Planning Constraints into Timed Automata. In *TIME-01. The 8th Int. Symposium on Temporal Representation and Reasoning*, pages 21–27, 2001.
19. 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.
20. O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In *STACS*, LNCS, pages 229–242. Springer, 1995.
21. T. Menzies and C. Pecheur. Verification and Validation and Artificial Intelligence. *Advances in Computers*, 65:5–45, 2005.
22. P. H. Morris and N. Muscettola. Temporal Dynamic Controllability Revisited. In *Proc. of AAAI 2005*, pages 1193–1198, 2005.
23. N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.
24. N. Muscettola, S. Smith, A. Cesta, and D. D'Aloisi. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems*, 12(1):28–37, 1992.
25. D. Smith, J. Frank, and A. Jonsson. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.
26. T. Vidal and H. Fargier. Handling Contingency in Temporal Constraint Networks: From Consistency To Controllabilities. *JETAI*, 11(1):23–45, 1999.