A Knowledge Engineering Environment for P&S with Timelines

Giulio Bernardi, Amedeo Cesta, Andrea Orlandini
CNR – National Research Council of Italy
Institute for Cognitive Science and Technology
Rome, Italy – {name.surname}@istc.cnr.it

Abstract
This paper presents some of the features of a knowledge engineering environment, called KEEN, created to support a timeline based planning based on the APsi-TRF modeling assumptions. A key feature of the environment is the integration of typical tools for knowledge based modeling and refining with services for validation and verification specialized to planning with timelines.

Introduction
Planning and Scheduling (P&S) systems have been deployed in several application domains. Most of these results have been achieved by small group of specialists molding their own specialized know-how. A key objective pursued in the P&S Knowledge Engineering sub community is the synthesis of software environments that would allow the development of applications to people that, as a minimum, are not “leading edge” specialist. Example of such environments are ITSIMPLE (Vaquero et al. 2013), GIPO (Simpson, Kitchin, and McCluskey 2007b), and EUROPA (Barreiro et al. 2012).

Over the last year and a half we have been developing our own Knowledge Engineering Environment (KEEN) that is built around the state of the art framework for P&S with timelines called APsi-TRF\(^1\) (Cesta et al. 2009). The particular perspective we are pursuing with KEEN is the one of integrating classical knowledge engineering features connected to support for domain definition, domain refinement, etc. with services of automated Validation and Verification (V&V) techniques as those surveyed in (Cesta et al. 2011). The current paper describes aspects of the current environment. In particular we describe new features of the environment for Domain Definition and Visualization and then some of the V&V tools at work starting from the defined domain. To make the example more concrete we have used as a running example the GOAC domain where we have accumulated quite an amount of basic knowledge.

The paper is organized as follows: a section describes basic knowledge on timelines to set the context and shortly introduces the GOAC domain, then the comprehensive idea of the KEEN system is described. Two following sections are dedicated to the functionalities for domain definition and to knowledge engineering services based on V&V. Related works and conclusions end the paper.

Timeline-based Planning
The main modeling assumption underlying the timeline-based approach (Muscettola 1994) is inspired by the classical Control Theory: the problem is modeled by identifying a set of relevant features whose temporal evolutions need to be controlled to obtain a desired behavior. In this respect, the set of domain features under control are modeled as a set of temporal functions whose values have to be decided over a time horizon. Such functions are synthesized during prob-

- 1\(^{\text{APsi-TRF is a tool of the European Space Agency (ESA) initially designed and built by our CNR group during the Advanced Planning and Scheduling Initiative (APSI).}}\)
lem solving by posting planning decisions. The evolution of a single temporal feature over a time horizon is called the timeline of that feature.

The timeline-based planning is an approach to temporal planning which has been applied to the solution of several space planning problems—e.g., (Muscettola 1994; Jonsson et al. 2000; Smith, Frank, and Jonsson 2000; Frank and Jonsson 2003; Chien et al. 2010). 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.

In this regard, we consider multi-valued state variables representing time varying features as defined in (Muscettola 1994; Cesta and Oddi 1996). As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal evolutions of timelines. For the state variables, such causal laws are encoded in a Domain Theory which determines the operational constraints of a given domain. Task of a planner is to find a sequence of control decisions that bring the variables into a final set of desired evolutions (i.e., the Planning Goals)

GOAC: a test planning domain

This work considers as running example a real world planning domain derived from a project funded by the European Space Agency (ESA). In fact, the Goal Oriented Autonomous Controller project (Ceballos et al. 2011) was an 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 (Cesta et al. 2009) and an executive a la T-REX (Py, Rajan, and McGann 2010); (b) a functional layer which integrates G^en^M and BIP (Bensalem et al. 2010).

Such robotic domain considers 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 (pan, tilt) = (0, 0). Finally, during the mission, the Orbiter may not be 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 motionless 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 motionless; (C5) While communicating, the orbiter has to be visible.

Timeline-based specification. 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.

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. 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 (Att(x,y)), moving towards a destination (GoingTo(x,y)) or Stuck (StuckAt(x,y))3. 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 ⟨x, y⟩ with the PTU in ⟨pan, tilt⟩ and store it as a file in the on-board memory (TakingPicture(file-id,x,y,pan,tilt)) or be idle (CamIdle()).

3 Sometimes, the robot may be stuck in a certain position and the navigation module should be reset.
Similarly, the communication facility can be operative and dumping a given file (Communicating(file-id)) or be idle (ComIdle()).

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).

![Diagram](image.png)

**Figure 2:** An example of timeline-based plan.

Fig. 2 exemplifies the use of synchronizations implementing the operative rules 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). 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).

In the actual domain model, an additional state variable is considered: the Mission Timeline. Such state variable is used just to model the reception from the external facilities of high level mission goals, i.e., TakePicture(pic, x, y, pan, tilt) and At(x,y) to model, respectively, the goal of taking a picture with a particular position/PTU setting and just moving the rover to a certain position. These goals are set on the Camera and RobotBase timeline as actual planning goals.

**The KEEN System**

As explained in (Cesta et al. 2009) the APSI-TRF environment is a development environment that gives “a timeline-based support” for modeling a domain. Its sketchy representation is the core of Figure 3, where it is accessible through a Domain Description Language and a Problem Description Language (the timeline equivalent of analogous files in classical planning) and it has a software machinery (the Component-Based Modeling Engine) that essentially produces a data structure here sketched as “current plan” that indeed is a Decision Network in TRF terminology (Cesta et al. 2009) that is a richer representation for representing the domain, the current problem, the law to achieve a solution, and the a flexible temporal plan at the end of a problem solving session. The APSI-TRF has capabilities for plugging in different problem solvers, also more than one for the same problem. For the current purposes we are solving GOAC problems with an APSI-compliant version of OMP5 (Fratini, Pecora, and Cesta 2008). In KEEN, the APSI-TRF is surrounded by a set of active services that give support during the knowledge engineering (KE) phase. Indeed in our view the knowledge engineering phase is interpreted in a very broad sense. For example we also have a Plan Execution block that contains a Dispatch Service to send actual commands to a controlled system and an Execution Feedback module that allows to receive the telemetry from an actual plan execution environment. The idea pursued is that you can connect the KEEN to an accurate simulator of the real environment, to a real physical system (e.g., a robot) and have functionalities to monitor with visual tools also the execution phase. We see in Figure 3 how KEEN is composed by “classical tools” you expect in a KE environments and by V&V services.

![Diagram](image.png)

**Figure 3:** The Knowledge Engineering ENvironment (KEEN) Design Support System.

In particular we here describe the Domain Editing and Visualization module that provides initial solution for a user interaction functionality for creating planning domain models. In this respect, we have developed an Eclipse plugin that provides a graphical interface to model, visualize and analyze the P&S domains. Additionally, plans can be generated by means of OMPS in a continuous loop of usage. The V&V services, comes from work described in papers like (Cesta et al. 2010b; 2010a; 2011; Orlandini et al. 2011b). They are all based on the use of Timed Game Automata, exploiting UPPAAL-TIGA (Behrmann et al. 2007). As a consequence their entry point is the TGA Encoding module that implements a translation from P&S specification to TGA. The other services rely on that encoding. The Domain Validation module is to support the model building activity providing a tool to assess the quality of the P&S models with respect to system requirements. Similarly, the Planner Validation module is also deputed to assess the P&S solver.
with respect to given requirements. But it is worth specifying that 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 and controller synthesis module is to check whether proposed solution plans are suitable for actual execution as well as to generate robust plan controllers. To implement the modules functionalities, verification tasks are performed by means of UPPAAL-TIGA. Such a tool extends UPPAAL (Larsen, Pettersson, and Yi 1997) providing a toolbox for the specification, simulation, and verification of real-time games. As a result, UPPAAL-TIGA is an additional core engine for KEEN.

Supporting Domain Definition
The most recent work concerning KEEN has concerned the support to timeline-based domain definition. Around this problem we have a first combination of “classical” KE tools and V&V services.

Bidirectional Editing and Visualization
Our goal for KEEN is to provide an integrated environment where the user may work both visually and at the traditional code level, while having the opportunity to easily verify and validate his/her work. A knowledge engineering environment like this is a complex piece of software, and it makes no sense to reinvent the wheel: some of the features we needed are standard and already supported by state-of-the-art development tools.

For this reason, KEEN is implemented as a plugin inside Eclipse® platform. Eclipse is one of the most widespread Integrated Development Environments (IDE) for many programming languages, Java above all. It provides a lot of features that are nowadays required for a professional IDE, like syntax highlighting, content assist, inline documentation, strong refactoring support, near real time compilation and code browsing, debugging, testing and integration with external tools, and many others. The Eclipse Platform can be extended by the means of plugins, thus providing a powerful environment for the implementors of new languages, who can leverage Eclipse’s key strengths to suit their needs.

KEEN uses standard Eclipse components to provide traditional code-level functionalities:
- A syntax highlighter, which uses different colors for different parts of the code to emphasize language keywords, special types, parameters, literals and so on. The central part of Figure 4 shows the code editor performing syntax highlighting.
- A tree view (Outline view) of relevant code blocks like state variables, providing a fast visualization and navigation inside the source files. In Figure 4, in the lower left corner, the Outline view is showing the GOAC Domain state variables: RobotBase, Platine, Camera, Communication, MissionTimeline, CommunicationVW.
- Real time syntax checks to easily spot erroneous programming constructs.

As before, the heart of KEEN is represented by the APSi framework, which, among other things, is used to maintain an updated representation of the problem being worked on. The use of APSi implies that KEEN is very loosely coupled to the particular language being used for Domain or Problem definition: while some Eclipse components (e.g. the syntax highlighter) are implemented for a specific language, their implementation is trivial and often consists in writing a grammar description file and not much more. But the more advanced features of KEEN are built on top of the APSi framework, so that concepts like state variables, timelines, or synchronizations are presented without an explicit dependency on the particular language the code is written in.

At the moment of writing, KEEN is endowed with a graphical representation of the Domain Model, as shown in Figure 5. Other graphical representations, like an execution-time timeline view, are being worked on. In this view, the state variables of the Domain Model are represented on a workbench by colored blocks, which can be moved around and expanded or collapsed to show/hide their values and constraints. Also, the desired state variables can be selected to show their synchronization relations with other state variables, represented by dotted lines between the blocks. The graphical representation has been designed to be as less tied to a particular graphical framework as possible, relying only on standard Java’s AWT libraries: this allows the environment to be used inside Eclipse, but has the potentiality of being reused in ad-hoc applications too.

In Figure 5, the six state variables of the GOAC Domain are shown: some of them have been expanded (Platine, Camera, MissionTimeline), while the others are collapsed (RobotBase, Communication, CommunicationVW). The currently selected variable, MissionTimeline, shows its synchronizations (depicted as arrows) with other three state variables: one from MissionTimeline.TakingPicture() to Camera.TakingPicture(), one from MissionTimeline.TakingPicture() to an unspecified value of Communication (since Communication state variable is col-

---

4http://www.eclipse.org
lapsed and its values are not shown), and one from Mission-Timeline.Art() to an, again, unspecified value of RobotBase.

The user may also use this graphical environment to define new state variables (right-clicking on an empty workbench area) and to add and edit their properties and values. The environment draws its information from the APSI framework, and immediately updates the APSI representation when the user makes a change.

At the end of the chain, when the internal representation of the model is changed, a language-specific component is used to trigger source code modifications, using Eclipse’s support for code refactoring. This way the tool allows the user to perform round-trip engineering\(^\text{3}\) by synchronizing source code and graphical views: the user can start to define a new model graphically, then switch to the traditional mode and do some hand-made editing, then switch back to the graphical mode and so on.

The integration of traditional IDE features and visual modeling functionalities should help both the experienced domain coder and the beginner or occasional writer: the former will probably use the traditional mode for the most part of its work, switching to the graphical mode to observe the results of its coding, while the latter might feel more comfortable in designing visually, switching to the code view to learn the language and experiment with it.

When the user is satisfied by the model, he/she can ask KEEN to generate a solution plan. Currently, KEEN does this by the means of the OMPS planner, but different planners will be added in the future. As for the case of the domain the plan representation is completely handled by APSI and a specific language generation component is deputed to the creation of a source file encoded with a Problem Description Language syntax. The user can then modify the generated solution plan at his/her will, and ask KEEN to perform plan verification using UPPAAL-TIGA (see later for further details). At the time of writing, KEEN allows the user to inspect and modify the generated plan in its textual form. In the future, a specialized plan editor similar to the one used for domain modeling will be added.

**Integrating V&V Services**

The deployment of formal methods techniques is to enhance the KEEN system with suitable V&V capabilities, thus constructing one of the main advantages in its use. In this regard, the KEEN takes advantage from a set of research results based on Timed Game Automata model checking (Cesta et al. 2010a; 2011; Orlandini et al. 2011b) to provide support over all the design and development cycle of P&S application with the APSI-TRF.

**Timed Game Automata** (Maler, Pnueli, and Sifakis 1995) (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 approach presented in (Cesta et al. 2010a), flexible timeline-based plan verification can be performed by solving a Reachability Game using UPPAAL-TIGA. To this end, flexible timeline-based plans, state variables, and domain theory descriptions are compiled as a set of TGA (nTGA): (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, a one-to-one mapping is defined 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, two locations are considered: 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 \(\cup\) \(\{A_{obs}\}\) 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. The encoding \(\mathcal{PL}\) is considered as the basis for implementing the V&V functionalities discussed in the following.

**Domain Validation.** Similarly to (Khatib, Muscettola, and Havelund 2001), the TGA encoding \(\mathcal{PL}\) can be exploited in order to validate planning domains, i.e., checking properties that are useful for ensuring correctness as well as detecting inconsistencies and flaws in the domain specification. For instance, undesired behaviors or safety properties can be checked against the planning model in order to guarantee the validity of the specification. 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 require-
ments\textsuperscript{6}.

To implement such a functionality, deriving from $\mathcal{PL}$ the nTGA $Dom = StateVar \cup \{A_{Obs}\}$, representing the allowed behaviors described by the associated planning domain, and stating a suitable Computation Tree Logic (CTL) formula $\phi$, representing a given system property $F$ to be checked, verifying $\phi$ in $Dom$ by means of UPPAAL-TIGA corresponds to validate the planning domain with respect to the property $F$.

Among relevant properties, values reachability is an important aspect that can be checked. Namely, the reachability of a value stated in the planning domain is checked starting from one specific initial state (or from each possible initial state). In this regard, the KEEN environment allows to perform a full reachability test for all the values declared in the domain and, in Fig. 6, the result for the GOAC domain is depicted. In particular, all the stated values are reachable except the $StuckAt$ in the navigation state variable that, obviously, cannot be planned but only detected as an abnormal system behavior. In general, finding that a certain value is unreachable may suggest either the presence of incomplete specifications or that some parts of the model are actually needed.

![Figure 6: Detail on the Domain Validation frame reporting results of the reachability test for all the allowed domain values.](image)

Also, the KEEN system allows to define user-defined properties to be checked (e.g., undesired or safety properties). For instance, a GOAC user may want to check that the $Communication$ value is always reachable after a $TakingPicture$. Having this property satisfied would confirm that the model allows to correctly manage the downlink actions for stored science pictures. This corresponds to check the following formula: $A\Box Camera.TakingPicture(file\_id=x,\ldots) \rightarrow E\Diamond Communication.Communicating(file\_id=x)$.

Another relevant property the user may check through the KEEN system is the violation of mutual exclusion for timeline’s allowed values. In fact, such test is useful for detecting an incomplete specification of synchronizations in the planning domain theory. For instance, in the case of a flawed GOAC domain, 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, could be verified, then, providing an evidence that the (C4) domain constraints might be violated.

**Planner validation.** In order to validate the planner, we are interested in checking that the planning solver works properly. In this sense, the application design activity should be supported by providing effective methods to validate the solver and the generated solutions, i.e., assessing its capability of generating a correct plan and, in addition, also the quality of the generated solution plans should be checked. For this purpose, the KEEN system has been endowed with two important submodules: 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. The KEEN Plan Verification and Plan Validation modules have been implemented exploiting the verification method presented in (Cesta et al. 2010a), i.e., solving the Reachability Game $RG(\mathcal{PL}, Init, Safe, Goal)$ defined as above.

**Plan Verification and Dynamic Controllability Check.** The Plan Verification module is fully relying on UPPAAL-TIGA by winning the Reachability Game $RG(\mathcal{PL}, Init, Safe, Goal)$. Then, the KEEN system invokes UPPAAL-TIGA for checking the CTL formula $\Phi = A[Safe \ U Goal]$ in $\mathcal{PL}$. In fact, the formula $\Phi$ states that along all its possible temporal evolutions, $\mathcal{PL}$ remains in Safe states until Goal states are reached. That is, in all the possible temporal evolutions of the timeline-based plan all the constraints and the plan is completed. 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 Fig. 7, a plan for the GOAC domain is verified and the system reports about its correctness taking advantage of the UPPAAL-TIGA verification process. Also, the dynamic controllability (Morris, Muscettola, and Vidal 2001) is checked and, in this case, successfully verified.

The feasibility of such method has been shown in (Cesta et al. 2010a; Orlandini et al. 2011b) where the verification methodology has been applied in two real-world planning domains.

**Plan validation.** Besides synchronization constraints, users 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 preferences on the global behavior of the generated plan. These requirements may be 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 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.

Examples of such relaxed constraints in the robotic case study may be no unnecessary tasks have been planned (e.g., unnecessary robot navigation tasks). This validation task results as an important step in assessing the plan quality as well as the planner effectiveness.

**Plan Controllers Synthesis.** Plans synthesized by temporal P&S systems may be temporally flexible hence they identify an envelope of possible solutions aimed at facing uncertainty during actual execution. In this regard, a valid plan can be brittle at execution time due to environment conditions that cannot be modeled in advance (e.g., disturbances). Previous works have tackled these issues within a Constraint-based Temporal Planning (CBTP) framework deploying specialized techniques based on temporal-constraint networks. Several authors (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005; Shah and Williams 2008) 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. Some recent works have addressed aspects of plan execution extending the approach in (Cesta et al. 2010a) by presenting the formal synthesis of a plan controller associated to a flexible temporal plan (Orlandini et al. 2011b). In particular, UPPAAL-TIGA is exploited in order to synthesize a robust execution controller of flexible temporal plans. In Figure 8, the execution strategy generated by UPPAAL-TIGA for a GOAC plan is shown by the KEEN interface.

**Related Works and Conclusions**

As said in the introduction, there are only a few general purpose KE tools for planning: GIPO (Simpson, Kitchin, and McCluskey 2007a), IT\$\$\$\$ (Vaquero et al. 2013), and EUROPA (Barreiro et al. 2012). Several tools do exist that address specialized aspects, nevertheless these three systems are the reference ones.

Most of the existing work has been dedicated to classical PDDL underlying language (this is the case for IT\$\$\$\$ and to some extent for GIPO). EUROPA has been till now the only timeline-based developing environment endowed with KE features.

With respect to both EUROPA and other research we are pursuing some distinctive features. For example the round-trip engineering functionalities in KEEN are rather new. While some of the existing systems can export to PDDL, and sometimes also allow the user to edit the produced PDDL file (as in IT\$\$\$\$), they do not support an integrated work practice in which the users can seamlessly switch between graphical and code views while maintaining the consistency between both views.

Standalone validation tools like VAL (Howey, Long, and Fox 2004) for PDDL language do exist, and are used by integrated environments to perform validation as in ModPlan (Edelkamp and Mehler 2005). Systems like GIPO and IT\$\$\$\$ do support static and dynamic analysis of the domains. The dynamic analysis though is performed by means of manual steppers (for GIPO) or simulation through Petri Nets (for IT\$\$\$\$). IT\$\$\$\$ also supports plan analysis by simulation.

Nevertheless, it is worth underscoring that simulation is not the same as the formal validation and verification proposed in KEEN. Somehow KEEN aims at filling a hole in existing knowledge engineering tools and nicely contribute to the whole picture.

Clearly there are other aspects, for example referring to those covered in the survey (Vaquero, Silva, and Beck 2011), that are still not address in KEEN. They will deserve specific work in the future.

**Acknowledgments.** Authors are supported by CNR under the GECKO Project (Progetto Bandiera “La Fabbrica del Futuro”).

*Figure 7:* The KEEN system showing the textual description of a plan for the GOAC domain. The pop-up window reports the result of the UPPAAL-TIGA verification. The plan is correct and, also, it results dynamically controllable.

*Figure 8:* The execution strategy generated by UPPAAL-TIGA is currently reported in the KEEN interface.
References


Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itsimple: Towards an integrated design system for real planning applications. The Knowledge Engineering Review.