Building a Domain-Independent Architecture for Planning, Learning and Execution. PELEA

César Guzmán
Universidad Politécnica de Valencia
cguzman@dsic.upv.es

Vidal Alcázar
Universidad Carlos III de Madrid
valcazar@inf.uc3m.es

David Prior
Universidad de Granada
dprior@decsai.ugr.es

Eva Onaindía
Universidad Politécnica de Valencia
onaindia@dsic.upv.es

Daniel Borrajo
Universidad Carlos III de Madrid
dborrajo@ia.uc3m.es

Juan Fdez-Olivares
Universidad de Granada, Granada
faro@decsai.ugr.es

Introduction

In this work, we present our ongoing effort on building a domain-independent software platform that integrates basic capabilities for planning, execution, monitoring, re-planning and learning. We name it PELEA after Planning, Execution and LEarning Architecture. The goal is two-fold: first, to provide software engineers a tool that can be used off-the-shelf to easily build planning applications, supporting a rapid prototyping life-cycle; and second to provide planning practitioners a tool that can be highly configured and in which new components replacing the ones that are already integrated can be easily added. Regarding the first goal, the platform currently includes state-of-the-art components for performing a wide range of (meta-)planning tasks, such as: planning (using several paradigms), controlled execution, monitoring of correct plan execution, re-planning when needed, learning of control knowledge, or low-level planning. Ultimately the user could use the tool as-is by giving as input a domain and problem descriptions. Regarding the second goal, it can serve as a benchmark platform for comparing different techniques under the same conditions. For example, a planning expert might want to try out a new re-planning technique on a robot simulator without the need to generate a complete planning-execution-monitoring-replanning architecture. We are currently interfacing the platform with known simulators (videogames and robotic platforms) as well as developing new ones for specific domains (logistics) and even a domain-independent temporal stochastic simulator. We are using this first prototype to develop some applications, such as a robotic system controlled by classical planning and a logistics transportation system.

We are building on our combined previous experience on developing different kinds of applications, ranging from fire extinction (Fdez-Olivares et al. 2006), logistics (Flórez et al. 2011), satellites maintenance operations (Rodríguez-Moreno, Borrajo, and Meziat 2004), education (Garrido et al. In Press), tourism (Castillo et al. 2008), or data mining (Fernández et al. In Press), among many others. In all these cases, the process of developing the final application is an "ad-hoc" manual process that requires expertise and techniques on at least two fronts: domain and problem modeling; and selection and configuration of planning systems, together with the implementation of execution controllers, monitoring tools and re-planning techniques, as well as the optional use of learning components. There has been some work on the first task based on powerful modeling tools such as ITSIMPLE (Vaquero et al. 2009). ITSIMPLE allows defining different kinds of planning models, as well as running diverse planners to generate solutions. However, it does not support further execution, monitoring and re-planning of those plans. We propose in this paper a tool that automates those steps.

There has also been previous work that defines generic architectures used for different purposes. Examples can be found in space and robotics applications with platforms like Mapgen (Ai-Chang et al. 2004), APSI (Cesta et al. 2009), PRS (Georgeff and Lansky 1987), or IxTeT (Ghallab and Laruelle 1994). Usually these platforms have been designed for particular planning techniques, as timeline-based planning (Ai-Chang et al. 2004; Cesta et al. 2009; Ghallab and Laruelle 1994), hierarchical planning (Fdez-Olivares et al. 2006), or reactive controllers (Georgeff and Lansky 1987). The goal of the PELEA project is to build a component-based architecture able to perform planning, execution, monitoring and learning in an integrated way, in the context of PDDL-based and HTN-based planning and suitable for a wide range of planning problems.

Next, we define the architecture and its component modules. The architecture allows planning engineers to easily generate new applications that integrate all planning and execution capabilities by reusing and modifying the components. A second scientific advantage of PELEA is to allow researchers or practitioners to compare techniques related to that functionality. We provide a set of tools that implement different techniques for each module, so that users can choose among those. The paper describes the on-going work on this architecture.

Overview of PELEA Architecture

PELEA architecture includes components that allow the applications to dynamically integrate planning, execution, monitoring, replanning and learning techniques. In general, there are two main types of reasoning: high-level (mostly deliberative) and low-level (mostly reactive). This is common to most robotics applications and reflects the separation between a reactive component and a deliberative component. However, in our architecture, these are simply two planning levels. This offers two main advantages: both lev-

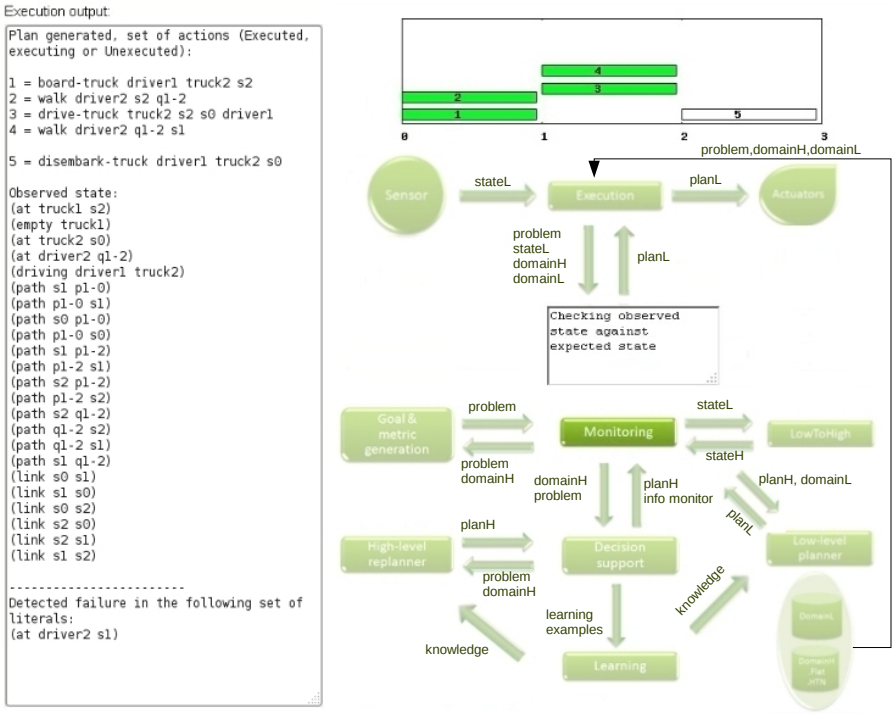

Figure 1: Screenshot of PELEA's web interface showing the architecture of the system. It shows the execution of a simple problem in the Driverlog domain.

els can be easily adapted to the requirements of the agent; and the differentiation allows the agent replanning at either level, which grants a greater degree of flexibility when recovering from failed executions. It would be possible to add additional levels to allow developers for a more hierarchical decision process. However, we consider that the sole distinction between high and low level is enough to tackle most problems, as has been shown in many robotics applications. Figure 1 shows a screenshot of PELEA's web interface and the current version of the architecture along with the integration of the modules. Even if we did not provide the explicit APIs, all modules in the architecture have access to either the high-level and low-level domain. We will describe in more detail later on the inputs and outputs of each component.

As we can see, PELEA is composed of eight modules that exchange a set of Knowledge Items (KI) during the reasoning and execution steps. The main KIs that we have used are (the modules also exchange the information related to the parameters that configure how each module works[1]):

- stateL: low-level state composed of the sensory information
- stateH: high-level state, translated from stateL as an aggregation or a generalization of low level information
- goals (problem): the set of high-level goals to be achieved by the architecture
- metrics (problem): the metrics that will be used in the high-level planning process
- planH: set of high level plans. Each high level plan is a set of actions resulting from the high-level planning process. The actions of these plans can also be the goals for the low-level planner (in case we want the low-level planner to act as a dynamic translation mechanism for high-level actions)
- planL: set of low level plans. Each low level plan is again an set of actions resulting from the low-level planning process. These actions should be operational, that is directly executable in the environment
- domainH: definition of actions for high-level planning
- domainL: definition of behaviors (skills) for low-level planning
- learning examples: to be used by the learning component to acquire knowledge for future planning episodes, either in the form of heuristics, domain models, or knowledge on the problem specification
- heuristics: in different forms (control rules, policies, cases, macro-actions, etc.) allow the planners to improve their efficiency in solving future planning episodes

[1]For instance, which planner to execute.

- info monitor: meta knowledge on the plan that helps to perform the monitoring (as, for instance, the generation time of a literal)

The PELEA architecture is controlled by a module, called Top-level control, which coordinates the execution and interaction of the Execution and Monitoring modules. As said above, PELEA architecture uses a two-level knowledge approach. The high-level knowledge describes general information, actions in terms of its preconditions and effects, and typically represents an abstraction of the real problem.

The high-level knowledge descriptions are rarely directly executable, if ever, they must be complemented by the low-level knowledge, which describes the more basic actions in the simulated world, and it is typically concerned with specific rather than general functions, and how they operate. The low-level knowledge is read from the environment through the sensors placed in the Execution module. The environment is either a hardware device, a software application, a software simulator, or a user. An example of low-level knowledge would be "the coordinates of a robot" or "degrees of motion of a robot arm". In PELEA, it is not necessary to work at the two knowledge levels. For instance, one can just work at the high-level, so that converting knowledge from high-level into low-level with the LowToHigh module or using the Low-level planner module are not needed. A more detailed description of the operation of the architecture of PELEA can be seen in (Alcázar et al. 2010). In the following, the life-cycle of the architecture is described.

Execution Module. The starting point of the architecture is the Execution module, which is initialized by the Top-level control, receiving a high-level and low-level domain, and a problem, composed of an initial state, a set of goals to achieve, a set of objects, and, optionally, a metric. The Execution is initialized with the domain and the problem, which in turn initializes the objects and their positions in the environment. The Execution keeps only the static part of the initial state, given that the dynamic part, called *stateL* (low-level state), will come from the environment through the sensors.

Monitoring Module. *stateL*, the problem and the domain are sent by the Top-level control to the Monitoring module to obtain a low-level plan (*planL*). The actions in *planL* are executed one by one by the Execution module (as can be seen in the Figure 1). As commented above, the modules LowToHigh and Low-level planner are only used in case the domain is modeled at the high and low levels. Otherwise, the Monitoring calls directly the Decision Support to obtain a high-level plan (*planH*). On the other hand, the module Goals&Metric Generation is invoked in case the problem goals or the metric change dynamically along the plan execution. Once the Monitoring module receives the necessary knowledge (state, problem and domain), it starts the monitoring process. The first step of the plan monitoring is to check whether the problem goals have already been achieved (*goalsL* and *goalsH* in case we are dealing with the two processes). If so, the plan execution finishes; otherwise, the Monitor begins with the first iteration of the plan monitoring.

Decision Support Module. At the first iteration of the algorithm, there is no plan to monitor yet, so the Monitoring calls the Decision Support, which obtains a valid plan that achieves the goals from the current observed state through the High-level replanner. This latter module receives a problem and a high-level domain (*domainH*), and generates a high-level plan (*planH*). *planH* is sent back to the Decision Support module, which computes the variables to be monitored and keeps this information in the parameter *info monitor*. Both *planH* and *info monitor* are sent by the Decision Support to the Monitoring.

Low-level Planner. The Monitoring module, with the help of the Low-level planner module, generates a set of executable low-level actions (*planL*), if this is the case. If the Low-level planner module is not being used, the Monitoring assumes that the high-level actions in *planH* are executable, and they are sent to the Execution module, which executes the actions one by one. Then, it senses the dynamic part of the state from the environment. The Monitoring receives the information from the observed state (*stateL*) after the execution of an action, and verifies the information in *stateL* against the parameter *info monitor*. If the values of all the checked variables are within the value range specified in *info monitor*, the Monitoring continues with the plan execution.

Replanning/Plan Repair. Otherwise, if a discrepancy between the expected and the observed state (*stateL*) is encountered, for instance, in the Figure 1 the Monitoring has detected a discrepancy in the literal *at driver2 s1*, which means that the action *walk driver2* has failed in the execution, the anomaly is reported by the Monitoring module to the Decision Support, which determines whether the discrepancy is relevant to the plan execution or not. That is, whether the plan is still valid to achieve the goals from the current observed state. At this point, the low-level planner can also be invoked to find the most immediate actions for a rapid intervention -if reactivity is needed- since this module typically stores predefined behaviours or courses of actions for reaching a situation. In case the Decision Support finds the anomaly entails a plan failure, and so the plan is no longer executable, it will take a decision about whether applying a plan repair, or replanning through the High-level replanner, thus starting a new iteration of the algorithm. Particularly, the Decision Support decides by an Anytime Plan-Adaptation approach (Garrido, Guzman, and Onainda 2010) whether it is worth repairing the plan, in which case it fixes *planH* and makes it executable again, or, it would be better to replan, in which case it requests a new plan to the High-level replanner module. In case that the discrepancy is not relevant to the plan validity, the Decision Support resumes the execution of *planH* by sending back the remaining and the new parameter *info monitor* to the Monitoring module, which in turn sends the next action to the Execution.

Whilst no discrepancies are found in the observed state, the two modules that are continuously interacting are the Monitoring and the Execution. The Monitoring not only checks for discrepancies but also if the problem goals (*goalsL* and *goalsH*) are already satisfied in the current state. In that case, the overall process is finished.

Currently, PELEA integrates, among others, with the following environments: Physical robot PIONEER 3DX through Player (robot independent platform for controlling robots of various kinds); Temporal probabilistic simulator, developed within the project that allows users to define temporal probabilistic domains, in the spirit of MDP-Sim (Younes and Littman 2004), for which we also have an API; Virtual Robot Simulator (VRS[2]) that is a freeware software suite for robotics applications; Alive (Fernández et al. 2008), an open platform for developing social and emotion oriented applications; and TIMI (Florez et al. 2010), a planning tool for real logistic problems.

Conclusions

In this paper, we have presented the ongoing work on building an architecture, PELEA, that integrates planning related processes, such as sensing, planning, execution, monitoring, replanning and learning. It is conceived as a flexible and modular architecture that can accommodate state-of-the-art techniques that are currently used in the whole process of planning. This kind of architectures will be a key resource to build new planning applications, where knowledge engineers will define some of the components, parametrize others, and reuse most of the available ones. This will allow engineers to easily and rapidly develop applications that incorporate planning capabilities. We believe this kind of architecture fills part of the technological gap between planning techniques and applications.

Acknowledgements

This work has been partially supported by the Spanish MICINN project TIN2008-06701-C03.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Alcázar, V.; Guzmán, C.; Milla, G.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindía, E. 2010. Pelea: Planning, learning and execution architecture. In *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodríguez, A.; Fernández, S.; Arias, J. D.; and Borrajo, D. 2008. SAMAP. A user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(2):1318–1332. ISSN: 0957-4174.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA*.

[2]http://robotica.isa.upv.es/virtualrobot/

Fdez-Olivares, J.; Castillo, L.; García-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together. experiences in SIADEX. In *Proc. ICAPS 2006*. Awarded as the Best Application Paper of this edition.

Fernández, S.; Asensio, J.; Jiménez, M.; and Borrajo, D. 2008. A social and emotional model for obtaining believable emergent behavior. In Traverso, P., and Pistore, M., eds., *Artificial Intelligence: Methodology, Systems, and Applications*, volume 5253/2008 of *Lecture Notes in Computer Science*, 395–399. Varna, Bulgaria: Springer Verlag.

Fernández, S.; de la Rosa, T.; Fernández, F.; Suárez, R.; Ortiz, J.; Borrajo, D.; and Manzano, D. In Press. Using automated planning for improving data mining processes. *Knowledge Engineering Review Journal*.

Florez, J. E.; García, J.; Álvaro Torralba; Linares, C.; Ángel Garcia-Olaya; and Borrajo, D. 2010. Timiplan: An application to solve multimodal transportation problems. In Steve Chien, G. C., and Yorke-Smith, N., eds., *Proceedings of the 2010 Scheduling and Planning Applications woRKshop (SPARK'10)*, 36–42.

Flórez, J. E.; Álvaro Torralba; García, J.; López, C. L.; Ángel García-Olaya; and Borrajo, D. 2011. Planning multi-modal transportation problems. In *Proceedings of ICAPS'11*. Freiburg (Germany): AAAI Press.

Garrido, A.; Onaindía, E.; Morales, L.; Castillo, L.; Fernández, S.; and Borrajo, D. In Press. On the automatic compilation of e-learning models to planning. *Knowledge Engineering Review Journal*.

Garrido, A.; Guzman, C.; and Onainda, E. 2010. Anytime plan-adaptation for continuous planning. In *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of AAAI-87 Sixth National Conference on Artificial Intelligence*, 677–68.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd International Conference on AI Planning Systems*.

Rodríguez-Moreno, M. D.; Borrajo, D.; and Meziat, D. 2004. An AI planning-based tool for scheduling satellite nominal operations. *AI Magazine* 25(4):9–27.

Vaquero, T.; Silva, J.; Ferreira, M.; Tonidandel, F.; and Beck, C. 2009. From requirements and analysis to pddl in itsimple3.0. In *Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling*.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School Of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.