A General-Purpose Architecture to Control Mobile Robots

Luis J. Manso[1], Luis V. Calderita[1], Pablo Bustos[1], Javier García[2], Moisés Martínez[2], Fernando Fernández[2], Adrián Romero-Garcés[3], and Antonio Bandera[3]

***Abstract*—Complex robotic tasks require the coordination of a considerable amount of skills. This is generally achieved generating and executing action plans that fulfill the preconditions of the given objective. These tasks can be highly dynamic, since the appearance of new objects or unexpected situations is a constant during the plan execution. In this context, robot control systems require the capability of managing a suitable world model (creating, removing or retyping dynamically objects as a result of the plan execution), and the capability of monitoring and replanning when unexpected situations are detected. In this paper we introduce a general-purpose architecture for autonomous mobile robots providing these features. The architecture allows to generate planning applications since it integrates planning, re-planning, monitoring and learning capabilities, and, at the same time, manages a consistent graph-like world model. Finally, we present some preliminary results of the deployment of such architecture in an advertisement promoting robot domain.**

***Index Terms*—cognitive architectures, autonomous robots, planning**

I. Introduction

Generally, the process of developing fully-functional robot control architectures requires expertise on techniques from several fields (*e.g.,* planning, controllers, learning), and the careful definition of the software architecture. However, in most of such applications, specifically tailored software had to be developed for the domain at hand, which usually lacks generality and reuse possibilities. Having a domain-independent architecture would reduce the requirement of expert knowledge and –derived from the code reuse– programming errors and the overall implementation time. However, the implementation of a general-purpose architecture is not trivial. It must provide the necessary technologies to support the requirements needed by all the robotic domains that such implementation is going to be used with.

One of the most relevant requirements of these architectures is the ability to provide a world model structure valid for multiple robotic domains. Such structure must support metric and symbolic properties and the ability to be used for planning, monitoring and learning purposes. It should also be dynamic, in the sense that the world model should support including, removing and modifying the elements in it. Other relevant requirement is to provide a long-term deliberative reasoning module based on Automated Planning [1], [2]. Automated planning is a problem-solving task that consists of obtaining a plan (set of instantiated actions), given a domain model (set of actions) and a problem (initial state and a set of goals). The plan execution transforms the initial state into a state where all goals are achieved. Such deliberative reasoning is being increasingly used in robotic systems

[1]Luis J. Manso , Luis Calderita, and P. Bustos are with Computer and Communication Technology Department, Universidad de Extremadura, Av. de la Universidad sn, 10071, Cáceres, Extremadura. lmanso@unex.es

[2]J. García, Moisés Martńez and Fernando Fernández are with Computer Science Department, Universidad Carlos III de Madrid, Av. de la Universidad 30, 28911, Leganés, Madrid, Spain.

[3]Adrián Romero-Garcés and Antonio Bandera are with Electronic Technology Department, Universidad de Málaga, Av. Cervantes 2, 29071, Málaga, Spain.

because it employs some form of forward projection (reasoning in depth about goals, preconditions, resources, and timing constraints) and provide reasonable responses in situations unforeseen by the designer [3], [4]. Building a reactive system can be a complex and time-consuming endeavor because of the need to pre-code all of the behaviors of the system for all foreseeable circumstances. However, deliberative reasoning often fail to guarantee a timely response in uncertainty or unexpected situations. Thus, a final requirement of the architecture is the ability to monitor the correct plan execution, resolving unexpected situations by re-planning, and to learn planning knowledge from the experience to reduce the response time in future unexpected situations. In this way, the architecture combines the deliberative reasoning and a more reactive one (learned from the deliberative) by exploiting the benefits of both kinds of methods: reasonable responses in unexpected situations and timely responses. In this paper we describe a general-purpose architecture to control robotic systems providing all these features.

There has been some previous work to design generic architectures. Examples can be found in space and robotics aplications of platforms as MAPGEN [5], PRS [6], or APSI [7]. However these architectures has been developed for specific problems and techniques as time-line based planning, hierarchical planning, or reactive controllers.

The remainder of the paper is organized as follows. Section II presents a detailed description of the proposed architecture. Section III presents the learning technique to learn plan-based policies. Section IV describes the general control flow of the architecture. Lastly, Section V provides some preliminary results in the deployment of the architecture in a mobile robot, and Section VI presents the conclusions and further research.

II. The Architecture

The architecture is depicted in Figure 1 and it is composed of three main elements: **a)** a high level module for planning, monitoring, and learning (composed by the modules within the red rectangle labeled as *PELEA*), **b)** an executive in charge of redirecting the plans from the planning module to the corresponding low-level modules and managing the representation of the world (composed by the elements within the blue rectangle labeled as *Executive*), and **c)** a set of domain-dependent modules which are in charge of performing the necessary physical and perceptual actions (the green rectangle labeled as *Domain-dependent modules*). The behavior of the domain-dependent modules (the ones that actually make the robot move and perceive its environment) is defined by the plan provided by the planner and the current world model, which is shared and cooperatively built by all the modules.

For planning, the robot uses PELEA (Planning and LEarning Architecture [8]), whose monitoring capabilities allows the system to decide if re-planning is necessary. The executive is in charge of retrieving the optimal action from PELEA, redirecting the actions to the low-level modules, and verifying that the representation of the world is consistent.

To enable representing symbolic and metric properties simultaneously the architecture works with a hybrid model, implemented as a graph of symbols which can be attributed with metric properties (see Figure 2). In the robot, these symbols represent the different elements that the robot needs to work with (*i.e.*, the different locations, the humans detected and their classification information, the panel, the robot, and its battery). The edges represent relationships between the symbols in the model. These edges are labeled depending on the nature of the relationship. Finally, depending on the type of each symbol, they are attributed with metric properties (*e.g.*, locations have x y z

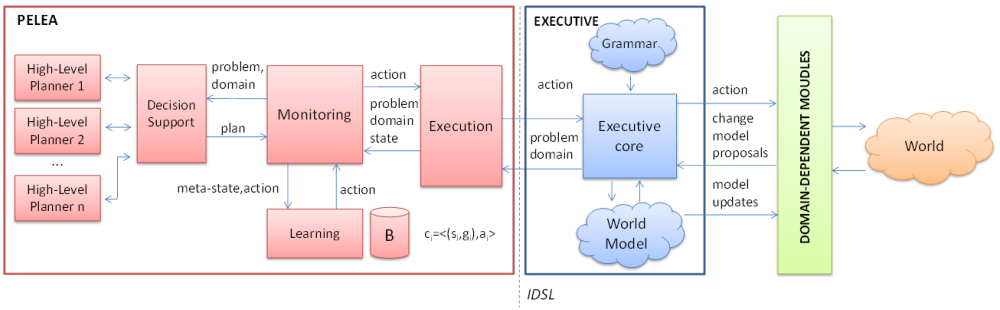

Fig. 1. Diagram of the software architecture proposed.

coordinates, the battery of the robot has a charge *level*). One of the advantages of this kind of hybrid representation is that it can be automatically converted to PDDL (standard representation language for planning domains [9]) by ignoring the metric properties, translating symbols to PDDL objects and links to PDDL predicates. If a metric property must be taken into account in the planning domain it can be translated to a symbol (of changing type depending on the value of the property) or to a predicate (in this case the value would be encoded in the label of the edge). More information regarding this kind of models and their advantages can be found in [10].

As a whole, the architecture works as an event-driven loop in which:

1) the executive asks PELEA for the optimal action (given the current world model and goal) and activates the domain-dependent modules according to the action
2) at some point a domain-dependent module executes an action or detects an unexpected event, and notifies the executive by proposing a change in the world model (including information about the event)
3) the executive verifies the change and, if valid, broadcasts the new world model to the rest of the modules (re-starting the loop).

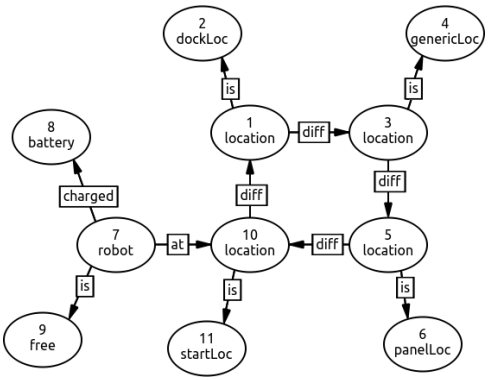

Fig. 2. Graphical representation of the initial model that the robot is given when started. It contains a symbol for the robot, its battery, its status, and four symbol pairs, one for each reference location (each symbol pair represents the location itself and an additional symbol used to specify the specific type of the location). When a human is detected the model is extended by including new symbols.

A. Planning, monitoring and learning

This section describes the planning, monitoring and learning system: PELEA [1]. It integrates planning, execution, monitoring, re-planning, and learning. As shown in Figure 1, PELEA is composed of four submodules that exchange a set of items during the execution steps. The main items that we have used are: *state*, abstracted high-level state (i.e., current world information); tuples *meta-state,action*, learning examples to be

[1]The PELEA architecture described here is an instantiation of the original version [8], where only some modules are used.

used by the learning component to acquire knowledge for future planning episodes; *domain*, definition of the model for high-level planning; *problem*, composed of the initial state and a set of goals to achieve; *plan*, a set of ordered actions resulting from the high-level planning process; *action*, a single piece of the plan. Now, each module of the architecture will be described.

1) Monitoring Module: The Monitoring module is the main module of PELEA. This is responsible for checking (monitoring) the plan progress during its execution. It is in charge of interacting with the Execution module by receiving a problem, a domain, and a *state* returning the next high level action to be executed. To do this, it uses the Decision Support module for a high level plan. Additionally, it is also in charge of receiving the current world information from the Execution module. When the Monitoring module receives the new world state, it will check the expected state against this perceived world state, *state*. If some of the monitored information do not fall within the expected range, the Monitoring will have to start another planning episode to compose a new plan according to new state perceived.

2) Execution Module: The Execution module is in charge of the interaction between PELEA and the environment. The environment can be either a software simulator (MDPSim), a hardware device (robot), a software application, or a user. In this work, the Execution module handles all the communication with the Executive by an IDSL Interface [11]. Particularly, it is responsible for initiating the work of PELEA by receiving a particular domain and problem to be solved and send the high-level actions to the Executive.

3) Decision Support Module: This module is in charge of receiving the domain and problem from the Monitoring module and returning a plan by the invocation of a high-level planner. Additionally, when the Monitoring informs about a discrepancy between the observed state and the expected planning state, the Decision Support also will invoke the high-level planner to update the plan to the new state. This module can be configured to use two different planners. First one is used for planning originally when there is no plan generated. Second one is used for re-planning. In this work, Fast-Downward [12] has been selected as planner and re-planner.

4) Learning module: It infers knowledge from the experience gathered by the high-level planner during the plan execution. The knowledge can be used either to update the domain planning model, to improve the planning process (for instance, learning heuristics), or to reduce the action response time by the learning of a plan-based policy as shown below in this paper.

B. Executive

The executive is the central element of the architecture but it is still simple and domain-independent. It activates the different low-level modules depending on the output of the planner and verifies that the changes they make to the current world model are valid. The domain of the robot is described as a set of graph rewriting rules equivalent to PDDL actions (with the exception that these rules can create and delete symbols). To enable communication with the planning system, the rules, the current world model and the robot's goal are transparently translated to PDDL before providing them to the planning system.

To ease the description and understanding of the planning rules by users which are not familiar with textual planning languages, the planning rules of the architecture are graph transformations which are specified graphically. Each rule is composed of two patterns, the one on the left-hand side (LHS) and the one on the right-hand side (RHS), and states that the model of the robot can be modified by substituting the LHS pattern with the one on the RHS. Figure 3 provides an example which describes that the robot can find new objects and generate plans involving such

action. When editing or visualizing a rule, the graphical editor automatically highlights in green those symbols that would be created by the rules, in red those that would be removed and, in gray, those symbols whose type would be changed by the execution of the rule.

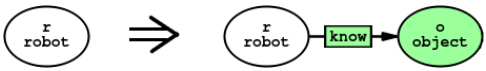

Fig. 3. Example of a graph transformation rule used to make the robot find new objects.

Besides asking PELEA for plans, the executive also verifies that the changes proposed by the domain-dependent modules are valid by posing the verification process as a planning problem, where the initial world is the current world and the goal is the new model proposed. Modifications are considered valid if and only if the planner can find a plan to get from the former to the later world model. This verification is performed as a planning process different of the main planning process conducted by PELEA. The objetive of PELEA is to provide a plan to reach the specified robot's goals (e.g., moving from a location A to a location B avoiding obstacles), while the verification planning process is used as a form of model checking [13].

III. Automated Plan-based Policy Learning

Since online planning and planning under uncertainty are both unrealistic for large-scale [14], we propose to follow an approach based on CBR (Case Base Reasoning) techniques to learn a plan-based policy, and, in this way, reduce the response time of PELEA when it faces unexpected situations. The policy learned is a mapping from the tuple $< state, goal >$ to action, $\pi : (S, G) \rightarrow A$. In this way, the policy, π, can be seen as an element that compiles a vast planning experience, in such a way that the agents can be used it to quickly make decisions. Traditional approaches learn these policies in a off-line way [15]: (i) sampling many instances of the stochastic problem, each instance being a challenging temporal and metric planning problem; (ii) solving each instance using a high-performing planner; and (iii) applying a classifier to learn the policy.

In this paper, we propose to learn such policy in an on-line manner using a different perspective with respect to the traditional ones. We illustrate our approach through the parent-child analogy, where a teacher takes the role of the parent and the learner agent takes the role of the child. At the beginning, the child does not know much about the world, hence, he/she needs to explore. While learning, most of the situations are new for him/her. In these unknown situations, he/she prefers to take actions advised by the parent. The child incorporates the parent's knowledge into his/her own knowledge about the world. After a while, the child feels comfortable about taking the actions learned from the parent. The unknown situations are become known situations. Translating this analogy into the PELEA architecture (Figure 1), the parent (teacher) corresponds to the Decission Support module which invokes the corresponding high-level planner, and the Learning module corresponds to the child.

A. Learning the Plan-based Policy

The policy $\pi : (S, G) \rightarrow A$ is represented as a Case Base, $B = \{c_1, c_2, \ldots, c_\eta\}$. Every case c_i consists of a pair state-goal $m_i = (s_i, g_i)$ and with the associated action a_i. Thus, $c_i = \{m_i, a_i\}$, where the first element represents the case's problem part and corresponds to the meta-state $m_i = (s_i, g_i)$, and the final element, a_i, depicts the case solution (i.e., the action expected when the agent is in the state s_i and wants to reach the goals g_i).

When the Monitoring module from PELEA receives a new meta-state, m_q, requests the learning module for an action. The learning module retrieves the nearest neighbor of

m_q in B, according to a given similarity metric and then returns the associated action. Traditionally, case-based approaches use a *density threshold*, θ, in order to determine when a new case should be added to the memory. When the distance of the nearest neighbor to m_q is greater than θ, a new case is added. In this sense, the parameter θ defines the size of the classification region for each case in B. If a new meta-state m_q is within the classification region of a case c_i, it is considered to be a known state. However, if it receives a meta-state m_q where the distance to any state in B is larger than θ, no case is retrieved. Consequently, the action to be performed from that state is unknown. In these cases, an action is required to the high-level planner (the teacher), and a new case is added to the case base B. In such a way, starting with an empty case-base, the learning algorithm continuously increases its competence by storing new experiences. Hence, the cases in B describe the plan-base policy, π, of the agent.

The online plan-based policy learning approach described here is similar in spirit to Case-based planning approaches (CBP) [16], [17]. In CBP, previously generated plans are stored as cases in memory and can be reused to solve similar planning problems in the future. However, the storing of full plans has two main drawbacks: a) the storing of long plans or plan created in complex dynamic environments can lead to increase memory consumption and response times [18], and b) in dynamic environments, with a high level of re-planning situations, the plan stored is rarely fully executed, only the first actions before a new unexpected situation happens.

B. An ad-hoc similarity metric for planning

To compute the distance between two meta-states, m_1 and m_2, let us assume that there are n predicates in m_1, $m_1 = \{p_1^1, , \ldots, p_n^1\}$, and m predicates in m_2, $m_2 = \{p_1^2, \ldots, p_m^2\}$. For each predicate $p \in m_1$ and $p \notin m_2$ and viceversa, the distance between the two meta-states, $d(m_1, m_2)$ is increased by 1. For each predicate, $p \in m_1$ and $p \in m_2$, we compute the distance between the set of literals, $P(m_1)$, of predicate $p \in m_1$, and the set of literals, $P(m_2)$, of predicate $p \in m_2$ as $d(m_1, m_2)$ $+= \sum_{k=1}^{K} \min_{p \in P(m_2)} d(p_k^1, p)$, where K is $|P(m_1)|$, and p_k^1 returns the literal kth from the set $P(m_1)$. Basically, this equation computes for each literal p_k^1, the minimal distance to every literals in $P(m_2)$. Finally, the distance between two literals of the same predicate is computed comparing the arguments as $d(p_l^1, p_h^2) = \frac{1}{J} \sum_{j=1}^{J} \delta(\arg_j p_l^1, \arg_j p_h^2)$, where J is the number of arguments, $\arg_j p_l^1$ returns the jth argument of literal p_l^1, $\arg_j p_h^2$ returns the jth argument of literal p_h^2, and δ returns 0 if both values are the same, and 1 if they are different.

IV. General Control Flow

The general control flow of the architecture is as follows:

1) Initially, the Executive composes the domain and the problem translating the starting world model and goals to PDDL. The Executive sends to PELEA the PDDL definitions of the domain and the initial problem.
2) The Execution module of PELEA receives the high-level domain and problem, This is composed of an initial state, a set of goals to achieve and a set of objects. The Execution sends the domain and problem to the Monitoring module.
3) Then, the loop is started:
    a) Monitoring checks whether it has achieved the goals. If so, it finishes. If not, it sends a request to the Learning Module with the state and the goals to be reached, $m_q = (s_q, g_q)$. The Learning Module retrieves from the base case B the case

$c_i = \{m_i = (s_i, g_i), a_i\}$ with $\min_i d(m_q, m_i)$:

i) If $\min_i d(m_q, m_i) \leq \theta$, the Monitoring sends to the Executive the action a_i.

ii) If $\min_i d(m_q, m_i) > \theta$, the Monitoring checks if exist an on-going plan:

- If so, and the plan can be continued, the Monitoring sends the next action, a, of the plan to the Execution. If not, the Monitoring calls to Decision Support module for re-planning. Then, Monitoring sends the first action, a, of the new plan to the Execution module.
- If not, the Monitoring calls to Decision Support for planning and sends the first action, a, of the plan generated to the Execution module.
- In any case, the Monitoring sends to the Learning Module the case $c = \{m_q = (s_q, g_q), a\}$ which is added to the case base, B, for future planning episodes.

b) The Execution module sends to the Executive the action received from the Monitoring.

c) The Executive sends the action to the corresponding domain-dependent module.

d) The Executive waits until the execution of the action has finished. Then, the Executive verifies the change model proposals and, if valid, it updates the world model.

e) The Executive sends to PELEA a new PDDL problem composed of the current world state, and a set of goals to achieve.

f) The Execution module of PELEA receives the problem and sends it to the Monitoring module and return to a).

V. Experimental results

The architecture has been tested using a promoting advertisement robot domain. In such domain, a robot is able to anticipate the person's trajectory in a public environment, and to provide product or service information [19], [20], [21]. Next, we describe in detail the domain, the graph grammar and some of the graph-rewriting rules used. Later, we provide experiments in the deployment of the architecture using a real Nomad robot, and finally, we test the learning capabilities of PELEA using a simulator.

A. Domain description

In this domain the robot is located in a specific location of a public environment (*e.g.*, shopping center, terminal airport) waiting to detect a human (we refer to such location as the *start area*). The objective of the robot is to offer passers-by to approach and interact with an advertising panel where products are promoted. Since there are different products, targeted for a wide variety of potential clients, the robot can select any person as potential customer. When it chooses a target, it moves from the *start area* to intersect the person's trajectory in order to engage interaction. This movement is short (3 meters maximum) and should allow the robot wait for the persons in a static pose, facing them and avoiding getting very close (1.5 meters minimum). At this point, the robot can say hello to the targets and avoid to scare them even if they are not very used to interact with an autonomous robot.

Once the interaction with the targets is engaged, the robot classifies them into a group (based on gender and age parameters) and will choose a *product topic* to offer. Product topics provide the robot a general theme to speak with people and to invite them to the *panel area*. While the robot tries to convince passers-by to approach the panel

the robot is always ready to say goodbye if they show intention of leaving the conversation or if the presented *product topic* does not seem to interest the selected person. On the other hand, the robot must also check its battery level to say goodbye and move to the *charging area* if its level is under a minimum value. The *charging area* is also close to the panel.

If the person agrees on going with the robot to the Panel area, the robot moves to the *panel area*, and there, it says goodbye. Then, it returns to the *starting area* and waits until the panel is unattended to start the whole process once again. As previously, if the battery level is under a safe value, the robot moves to the *charging area* to recharge.

B. The graph grammar and graph-rewriting rules

In addition to the first rule example, shown in Fig. 3, this section provides two rules used to demonstrate can rules be graphically described. The first rule, shown in Fig. 4, states that the robot can charge itself. The operations made by the rules are highlighted by the colors: in this case, the link between the robot and its battery, labeled as "discharged" is changed to "charged". While the rest of the model is left unchanged, the left-hand side of the rule also specifies other elements that must be found in the model in order to be able to execute the rule: the robot must be located in a position classified as a dock station. No symbols are created, deleted or modified by this rule, only edges.

The second rule, shown in Fig. 5, states that the robot can forget a person. In this case, there must be a detected person related to two additional symbols (of type productInfo and personInfo) which are deleted when the rule is executed. The symbol of the robot remains unchanged.

The previous rules are applied to the world model whenever a domain-dependent module requests a change model proposal, and this one is a valid change. The Executive sends the updated world model to PELEA which return the following action according to the current world situation.

C. Nomad experiments

The software architecture was implemented using the RoboComp framework [22] and has been tested on a Nomad 200 mobile robot from Nomadic [23]. Additionally, we have set a Windows Kinect on the top of the turret which supports movement, voice, and gesture recognition. PELEA uses Fastdownward [12] as the high-level planner for plan and replan when necessary. In this domain, we use three main domain-dependent modules: *navigation*, used to implement navigation actions; *person*, in charge of detecting, tracking and modeling the humans the robot interacts with; and *conversational*, used to understand and generate speech according to the interaction context. These domain-dependent modules are also developed as RoboComp components [22].

Figure 6 shows highlights of a video sequence in which the Nomad is performing the use case described in Section V-A. The first frame shows the Nomad robot in the start position, the potential consumer in front of the robot, and the panel. In this point, PELEA generates a plan and the robot executes the first two high-level actions: *searchPerson* and *detectPerson*. Once the potential consumer is correctly detected, the robot moves to the person. We can see this in the second frame of Figure 6 by the execution of the high-level action *goToPerson*. In the third frame, the robot is at social distance, and begins the interaction with the person. The robot greets the person (*sayHello*), and classifies it to establish the genre and an age range (*classifyPerson*). According to this classification, the robot chooses a product (*chooseProduct*), offers this product and captures the person interest (*capturePersonAttention*). In the last frame, the person is

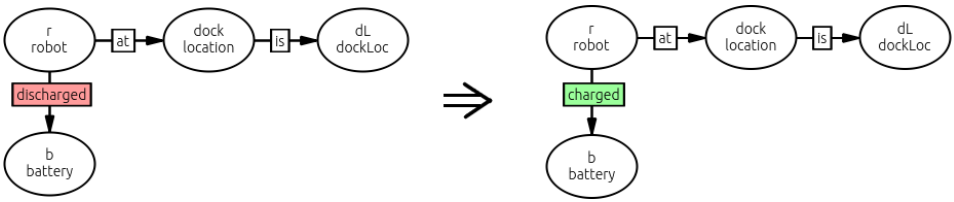

Fig. 4. Example rule used to charge the robot. It modifies the edge linking the robot and its battery from "discharged" to "charged".

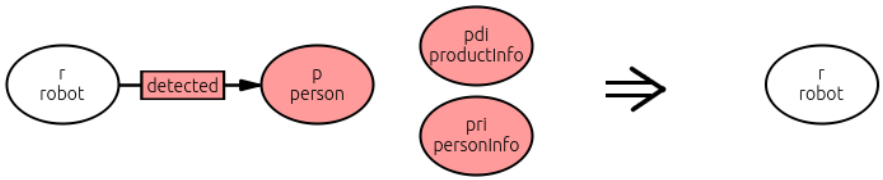

Fig. 5. Example rule used to forget a person. It removes the symbol of a person and two additional symbols which are related to the person.

interested and the robot drive it to the panel by the execution of the high-level action *goToPanel*.

D. Learning experiments

Next we provide additional experiments to validate the planning, re-planning and learning capabilities of PELEA. In these experiments, we have used the simulator MDPSim [24] to simulate the execution of plans (i.e., MDPSim replaces the Executive and the domain-dependent modules from the architecture in Figure 1). To simulate failures, we provide a probabilistic version of the domain in PPDDL to MDPSim, where actions can generate unexpected states with different probabilities. For each action there is a probability of $10-20\%$ the robot runs out of battery, and a probability of $10-20\%$ the robot looses the person. Furthermore, there is a probability of 30% the person is not interested in the product or service. Additional probabilities have been introduced to simulate unexpected states. The experiments are conducted on an Intel Dual-Core PC with 2 GB RAM and 2.4 GHz clock speed. In these first experiments, the learning module is disabled (i.e., PELEA always invokes the Decision Support to generate the initial plan and for re-planning unexpected situations). Figure 7 shows the response times of PELEA for five different planning processes.

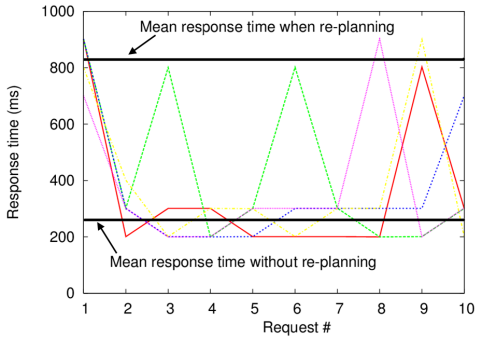

Fig. 7. Response times of PELEA for five different planning processes.

For each planning process, the x-axis shows ten consecutive action requests, and for each action request, the y-axis shows the time taken by PELEA to send the action to MDPSim. For each planning process, the first action request implies PELEA have to re-plan (or generate the initial plan). For the subsequents action requests there are two possibilities. If MDPSim sends to PELEA a valid state (Section *General Control Flow*), PELEA sends to MDPSim the next action

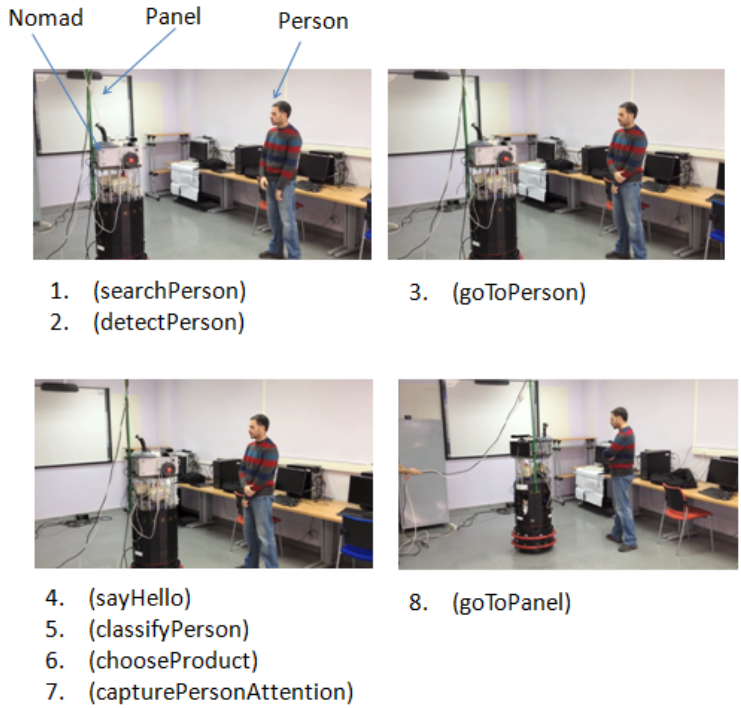

Fig. 6. Highlights of a video sequence with the Nomad and a potential consumer performing the basic use case. The PDDL actions performed are depicted below each frame.

of the plan without re-planning (in these cases, the mean response time is around 259 ms). Otherwise, PELEA replans and sends to MDPSim the first action of the new plan generated (the mean response time when re-planning is around 828 ms).

Next we enable the learning module of PELEA to validate the plan-based policy is correctly learned, and to measure the improvement of the response time using this policy in unexpected situations. Figure 8 shows five different planning processes and, for each one, the response times of the first fifteen action requests which require re-planning. In these experiments the θ is fixed to $\theta = 0$. At the beginning of the planning processes with an empty case-base B, the re-planning is performed in most of the cases by the Decision Support module. In these cases, the mean response time of PELEA is around 828 ms. As the learning process continues, new cases are added to B and the plan-based policy is learned. At around the requests 8-10, practically all steps are performed using the plan-based policy and Decision Support is rarely used. In these cases, the mean response time of PELEA is around 325 ms.

Therefore, the response time of the plan-based policy learned is slighly higher than the response time of PELEA with the learning module disabled when no re-planning is required (around 65 ms higher). However, as shown in Figure 8, the response time when re-planning is considerable reduced from 828 ms to around 325 ms.

VI. Conclusions

In this paper we have introduced a general-purpose software architecture to con-

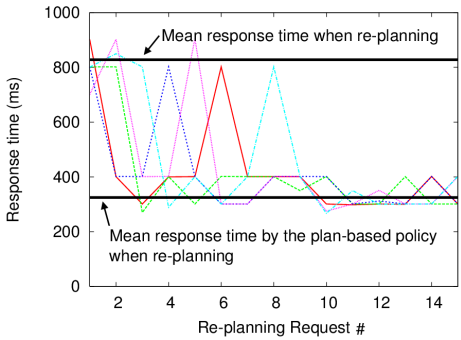

Fig. 8. Response times of the plan-based policy learned.

trol mobile robots. We successfully tested it as a robot control system in a Nomad robot using a promoting advertisement robot domain. However, the proposed architecture could be easily adapted to other robotic tasks by changing the domain-dependent modules, and the graph grammar. The graph grammar and the graph-rewriting rules let us efficiently represent dynamic environments, and create and destroy new objects (capabilities which are not present even in the newest extensions of PDDL). Additional experiments showed the planning, re-planning and learning capabilities of PELEA. The experiments with the learning capabilities disabled demonstre that, when no re-planning is required, PELEA's response times remain in a proper range for a correct social interaction. However, the response times increase when PELEA re-plans, which can be a nuisance for a person waiting for an answer. To mitigate this problem, we have proposed learn a plan-based policy from the experience gathered in on-line executions of the high-level planner. The experiments with the learning module enabled demonstrate PELEA correctly learns a plan-based policy in which response times remain in a proper range for both expected or unexpected situations. Learning the plan-based policy in an on-line way provides two main benefits over the off-line learning [15]: a) the plan-based policy is built progressively using the experience gathered in real executions, and b) avoids storing rare cases in the case base which are not used later in real executions. It is important to note that the benefits of the learning capabilities of PELEA shown in Section V-D result in a lower response time of the global architecture, and this lower response time is essential for a more suitable social interaction between the robot and the user.

As future work, we consider the deployment of the architecture in different robot domains, and exhaustive testing processes in both simulated and real environments.

ACKNOWLEDGMENT

This paper has been partially supported by the Spanish Ministerio de Economía y Competitividad TIN2012-TIN2012-38079 and FEDER funds, and by the Innterconecta Programme 2011 project ITC-20111030 ADAPTA.

REFERENCES

[1] C. Elsaesser and M. G. Slack, "Integrating deliberative planning in a robot architecture," in *NASA CONFERENCE PUBLICATION*, pp. 782–782, NASA, 1994.

[2] C. McGann, F. Py, K. Rajan, and A. Olaya, "Integrated Planning and Execution for Robotic Exploration," in *Intnl. Workshop on Hybrid Control of Autonomous Systems, in IJCAI09*, (Pasadena, California), 2009.

[3] R. A. Brooks, "Integrated systems based on behaviors," *Stanford University*, vol. 2, pp. 46–50, 1991.

[4] J. Blythe and W. S. Reilly, "Integrating reactive and deliberative planning in a household robot," in *In Proceedings of the aaai Fall Symposium on Instantiating Real-World Agents*, pp. 6–13, 1993.

[5] M. Ai-Chang, J. L. Bresina, L. Charest, A. Chase, and J. C. jung Hsu, "Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 8–12, 2004.

[6] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 2*, AAAI'87, pp. 677–682, AAAI Press, 1987.

[7] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi, "Developing an end-to-end planning application from a timeline representation framework," in *IAAI* (K. Z. Haigh and N. Rychtyckyj, eds.), AAAI, 2009.

[8] E. Quintero, V. Alcázar, D. Borrajo, J. Fernández-Olivares, F. Fernández, A. G. Olaya, C. Guzmán, E. Onaindia, and D. Prior, "Autonomous mobile robot control and learning with the pelea architecture.," in *Automated Action Planning for Autonomous Mobile Robots*, 2011.

[9] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The Planning Domain Definition Language," 1998.

[10] L. J. Manso, *Perception as Stochastic Sampling on Dynamic Graph Spaces*. PhD thesis, Cáceres Polytechnic School, University of Extremadura, 2013.

[11] A. Romero-Garcés, L. Manso, M. A. Gutierrez, R. Cintas, and P. Bustos, "Improving the lifecycle of robotics components using domain-specific languages," *CoRR*, 2013.

[12] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[13] *PDDL Planning Problems And GROOVE Graph Transformations: Combining Two Worlds With A Translator*, vol. 17, University of Twente, 2012.

[14] J. Garcia, J. E. Florez, A. Torralba, D. Borrajo, C. Linares Lopez, A. Garcia-Olaya, and J. Saenz, "Combining linear programming and automated planning to solve intermodal transportation problems," *European Journal of Operational Research (EJOR)*, vol. 227, pp. 216–226, 2013.

[15] M. Fox, D. Long, and D. Magazzeni, "Plan-based policy-learning for autonomous feature tracking," in *ICAPS* (L. McCluskey, B. Williams, J. R. Silva, and B. Bonet, eds.), AAAI, 2012.

[16] M. M. Veloso, H. Munoz-Avila, and R. Bergmann, "General-purpose case-based planning: Methods and systems," *AI Communications*, vol. 9, no. 3, pp. 128–137, 1996. Also in Kunstliche Intelligenz, 1:22-28, 1996, in German (with reversed order of the authors).

[17] L. Spalzzi, "A survey on case-based planning," *Artif. Intell. Rev.*, vol. 16, pp. 3–36, Sept. 2001.

[18] J. Schmitt, M. Roth, R. Kiefhaber, F. Kluge, and T. Ungerer, "Concept of a reflex manager to enhance the planner component of an autonomic/organic system," in *Proceedings of the 8th International Conference on Autonomic and Trusted Computing*, ATC'11, (Berlin, Heidelberg), pp. 19–30, Springer-Verlag, 2011.

[19] T. Kanda, D. F. Glas, M. Shiomi, H. Ishiguro, and N. Hagita, "Who will be the customer?: A social robot that anticipates people's behavior from their trajectories," in *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, (New York, NY, USA), pp. 380–389, ACM, 2008.

[20] T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita, "An affective guide robot in a shopping mall," in *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, HRI '09, (New York, NY, USA), pp. 173–180, ACM, 2009.

[21] H.-M. Gross, H. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley, "Toomas: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials," in *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'09, (Piscataway, NJ, USA), pp. 2005–2012, IEEE Press, 2009.

[22] L. J. Manso, P. Bachiller, P. Bustos, P. Núñez, R. Cintas, and L. Calderita, "RoboComp: a Tool-based Robotics Framework," in *Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, pp. 251–262, 2010.

[23] A. Chopra, M. Obsniuk, and M. R. Jenkin, "The nomad 200 and the nomad superscout: Reverse engineered and resurrected," *2013 International Conference on Computer and Robot Vision*, vol. 0, p. 55, 2006.

[24] H. L. S. Younes, M. L. Littman, D. Weissman, and J. Asmuth, "The first probabilistic track of the international planning competition," *J. Artif. Int. Res.*, vol. 24, pp. 851–887, Dec. 2005.