Blended Planning and Acting: Preliminary Approach, Research Challenges

Dana S. Nau
Department of Computer Science
and Institute for Systems Research
University of Maryland
College Park, MD, USA

Malik Ghallab
LAAS/CNRS
University of Toulouse
Toulouse, France

Paolo Traverso
FBK ICT IRST
Trento, Italy

Abstract

In a recent position paper in *Artificial Intelligence*, we argued that the automated planning research literature has underestimated the importance and difficulty of *deliberative acting*, which is more than just interleaving planning and execution. We called for more research on the AI problems that emerge when attempting to integrate acting with planning.

To provide a basis for such research, it will be important to have a formalization of acting that can be useful in practice. This is needed in the same way that a formal account of planning was necessary for research on planning. We describe some first steps toward developing such a formalization, and invite readers to carry out research along this line.

Introduction

Research on automated planning (the automatic generation of plans of action) has led to numerous impressive research accomplishments by an active community of researchers. Despite the large potential of these accomplishments, their deployment into fielded applications has unfortunately been relatively low. In (Ghallab, Nau, and Traverso 2014), we argued that the *acting* part is more important and more difficult than researchers have realized, and that more research is needed on the numerous AI problems that emerge when attempting to integrate acting with planning.

AI planning research has focused mostly on offline planning. Substantial work has been done on execution control (see the Related Work section); but most of it has treated actions as atomic. This underestimates the importance and difficulty of the deliberation needed to carry out the actions. More specifically:

- *Acting requires* ***continual online*** *planning and deliberation*. Throughout the acting process, an actor must refine its actions into lower-level steps, monitor and react to unexpected events, update and repair its plans. (see Fig. 1). Plans remain partial and abstract as long as the cost of possible mistakes is lower than the cost of modeling, information gathering, and thorough planning.
- *Actors typically are* ***heterogeneous hierarchies***: organized collections of heterogeneous modules for various specialized tasks. Planning for such an actor goes beyond existing hierarchical planning techniques; its requirements and scope are significantly different. Different modules may have different notions of what constitutes a state or an action, and what actions are available in what states. To integrate the modules effectively, the actor must have ways to translate among these representations.

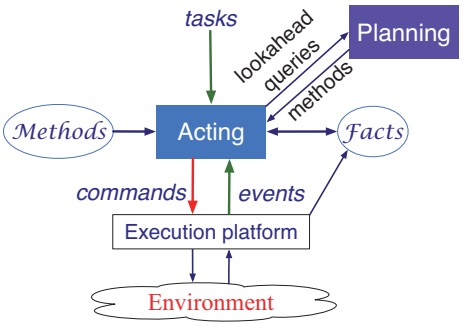

Figure 1: Acting with continual online planning.

This focus of this paper is the long-term challenge of developing a *formalization of acting that can be useful in practice.* Such a formal account is needed, in the same way that a formal account of planning was necessary for research on planning. As steps toward developing such an account, we propose preliminary versions of a hierarchical representation for both planning and acting, and acting and planning algorithms using this representation. We discuss the challenges that must be overcome to extend this work.

Related Work

Our ideas build on prior research by many authors including (Rosenschein and Kaelbling 1986), (Dean and Wellman 1991), and (Pollack and Horty 1999). These problems have been particularly of interest to the robotics community, e.g., (Despouys and Ingrand 1999; Ingham, Ragno, and Williams 2001; Kortenkamp and Simmons 2007; Effinger, Williams, and Hofmann 2010; Beetz, Mösenlechner, and Tenorth 2010) and the multi-agent community, e.g., (desJardins et al. 1999; Pappachan and Durfee 2000; Brenner and Nebel 2009).

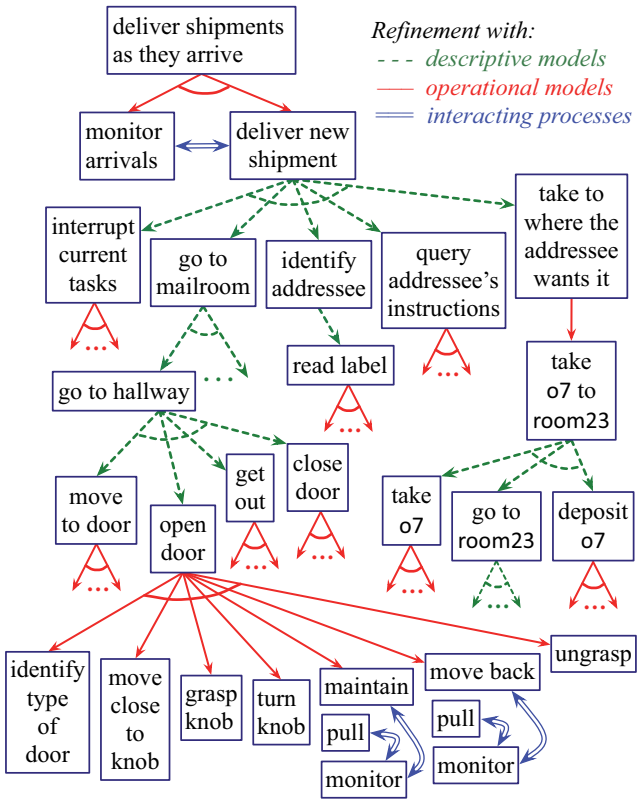

Figure 2: Tasks and actions for a service robot.

Starting with the early Planex (Fikes 1971), many systems considered planning primitives as directly executable; they focused on their execution control and monitoring. The lack of robustness of these systems was addressed by several approaches for specifying operational models of actions and for using these models to refine actions into lower level commands. Various representation have been proposed, such as procedures (e.g., RAP (Firby 1987), PRS (Ingrand et al. 1996), TCA (Simmons 1992), TDL (Simmons and Apfelbaum 1998)) transformation rules (e.g., XFRM (Beetz and McDermott 1994)), situation calculus (e.g., GOLEX (Hähnel, Burgard, and Lakemeyer 1998)), Petri nets and automata (PLEXIL (Verma et al. 2005), SMACH (Bohren et al. 2011), or (Wang et al. 1991; Barbier et al. 2006; Ziparo et al. 2011)). In some of these system, execution control is interleaved with planning (Löhr et al. 2012), including within a receding horizon framework (Garcia, Prett, and Morari 1989). Most of these systems introduce a separate knowledge representation between planning and acting; they have a clear separation between a planning stage and an acting stage, even when the two are interleaved.[1]

To the best of our knowledge, none of these systems addresses our objectives of a uniform framework for deliberation at the acting level, including planning, through a hierarchy of state and action spaces.

[1]Space limitation precludes a full discussion of the state of the art, more can be found in the survey of (Ingrand and Ghallab 2015).

Motivating Example

To illustrate some of the key problems, consider a versatile service robot working in an indoor environment such as a house or a store. The task is to fetch packages as they arrive, and bring them to the addressee. In Fig. 2, the boxes represent actions, and the red and green arrows denote refinement of abstract actions into collections of less-abstract actions.

Where there are dashed green lines, the action refinement could be done quite easily by a conventional planning algorithm. The action to be refined (e.g., "take o7 to room23") can be taken as the goal of the planning problem. To synthesize the sequence of actions ("take o7", "go to room23", "deposit o7"), most planning algorithms use **descriptive** action models (e.g., classical precondition-and-effects models) that describe *what* the action do, without describing *how* those actions will be carried out. But this does not mean that the robot would do that part of its deliberation offline, because the robot may not know until runtime what object should be taken where. For example, the addressee may want some objects to be left in his/her office and others to be delivered to him/her personally—and in the latter case, the robot needs to find out whether the addressee is currently in the office or somewhere else.

Where there are solid red lines, action refinement can more easily be performed using an **operational** model, which is basically a program telling *how* to perform action under certain circumstances. For example, near the bottom of the figure, opening the door depends on whether the door slides or turns, whether it opens to the right or to the left, whether it opens toward or away from the robot, what type of handle it has, etc. The robot will not learn this information until it goes to the door and looks at it; and it may not learn some other relevant information (e.g., whether the door is locked, whether it is jammed, whether it has a broken hinge) until it tries to open the door.

In a *closed* world where all of the possible effects of each action are known in advance, may be possible (at least in principle) for the robot to construct an offline plan for opening a door, using conventional techniques for planning in nondeterministic environments. But such an approach is difficult if the environment is dynamically changing and isn't pre-engineered to conform to the robot's needs; and it may not work at all if the action models or the models of exogenous events are incomplete.

The solid blue lines represent communications among interacting concurrent actions. For example, once the robot has turned the doorknob, it needs to simultaneously maintain the doorknob's turned position and pull backward to open the door, with continual monitoring to modify the actions in real time, to ensure that they are carried out correctly and have the desired effects. Descriptive action models are insufficient for this: the robot needs to synthesize the process and the interaction protocol. In some cases, this could be done by the automated synthesis of control automata which takes into account uncertainty and the interleaving of the execution of different processes at run-time (see, e.g., (Bertoli, Pistore, and Traverso 2010; Bucchiarone et al. 2013)). In other cases (e.g., the top-level decomposition in the figure), it would be more practical for

```
method m-opendoor(r, d, l, o)
 task:  opendoor(r, d)
  pre:  loc(r) = l ∧ adjacent(l, d) ∧ handle(d, o)
 body:  while ¬reachable(r, o) do
            move-closer(r, o)
            start-monitor-status(r, d)
            if door-status(d)=closed then
                unlatch(r, d)
            throw-wide(r, d)
            end-monitor-status(r, d)
```

Figure 3: A refinement method for the "open door" task in Fig. 2. The "pre" line gives the method's preconditions, and the body is a simple program to generate a task refinement.

a human to write a program in advance.

Unifying Descriptive and Operational Models

From the previous section, we discussed two kinds of action models: descriptive models like those used by in offline planning systems, and operational models like those used in reactive systems such as PRS (Ingrand et al. 1996)). If a single knowledge representation could be used for both, this would make it easier to specify and maintain consistency between the actor's reasoning and the planner's reasoning, thus making it more likely that planners will accurately predict the effects of actions and plans.

As a first step in this direction, we propose a *refinement method* representation. The refinement methods would be similar both to PRS procedures (Ingrand et al. 1996) and to SHOP's task-decomposition methods (Nau et al. 1999). As a suggestion of what such a representation might look like, let us say that a refinement method is a triple

$$m = (\mathit{role}, \mathit{precond}, \mathit{body})$$

that specifies a way to accomplish a *role* (which may be a task, event, or goal) by executing a simple program given in the method's *body*. For example, Fig. 3 gives a method for the opendoor task in Fig. 2. The steps in the body include control structures ("while" and "if"), commands to the execution platform, and tasks to be further refined.

If defined correctly, such a representation would ease the consistency problems mentioned above, because the same refinement methods could be used in both acting and planning algorithms. The next two sections outline what such algorithms might look like.

Research challenges. Developing such a representation—as well as the planning and acting algorithms to use it—will present several research challenges. For example:

- **Heterogenous hierarchy.** A hierarchically organized actor may use different state spaces and action spaces in different parts of its hierarchy. For example, in Fig. 2, room23 is an abstract entity that, at a lower level of the hierarchy, may correspond to a large set of possible configuration coordinates for the robot. A principled way is needed to compute mappings between lower-level information such as these configuration coordinates, and higher-level abstract entities such as room23. Research is needed on what kinds of mathematical and computational properties these mappings should satisfy, and how to incorporate these mappings into acting and planning algorithms.
- **Time.** Research is needed on how to generalize refinement methods over temporally qualified assertions, actions and tasks—and how to develop temporal planning and acting algorithms for such methods. Temporal Plan Networks of the RMPL system (Williams and Abramson 2001) offer an approach which extends temporal networks with symbolic constraints and decision nodes. Deliberation in such a system is finding a path in the explicit network that meets the local constraints, taking into account choices (Conrad, Shah, and Williams 2009) and possible error recovery (Effinger, Williams, and Hofmann 2010). More work for the synthesis of these networks from descriptive and operational specifications is needed.

Acting with Refinement Methods

The Refinement Acting Engine (RAE) in Fig. 4 is our first attempt at a formalism for acting, using refinement methods as operational models of actions. RAE is inspired by and formalizes the PRS system (Ingrand et al. 1996). It takes as input a stream of tasks to perform (which may come from a planner, a user, or RAE itself as updates of its own reasoning state). It delivers commands to the execution platform as required by the context given by a set of facts reflecting the perceived state of the world.

For each task τ in the input stream, RAE selects a relevant method m and creates a LIFO stack to keep track of how the refinement is progressing. If m fails, RAE will try a currently applicable alternative method that it has not already tried. This differs from backtracking, because since RAE may have executed actions while refining m.

Progressing in a stack σ means advancing sequentially by *one* step in the body of the topmost method. If this step is a command, RAE sends it to the execution platform; if the step is a task, RAE selects a relevant method and put it at the top of σ; if the step is to exit from m, RAE removes m from σ (and removes the stack completely if this makes σ empty).

While RAE is advancing on a stack, other tasks may appear in its input stream and require attention. Thus RAE will need to interleave the processing of all stacks.

In summary, the idea is for RAE to maintain a separate stack for each external task or event it reads in its input stream, and progress the stacks concurrently.

Research challenges. RAE is a simple refinement formalism for acting; additional work is needed to complete the formalism. In addition to the challenges mentioned in the previous section, here are some additional ones:

- The use of concurrent actions and concurrent tasks in RAE presents potential conflict issues (e.g., resource sharing). Currently, the only way to resolve such issues in RAE is to incorporate *ad hoc* fixes in the definitions of the methods.

```
RAE(M)
loop
   for each new task τ in the input stream do
      candidates ← {applicable methods for τ}
      if candidates = ∅ then output("τ failed")
      else do
         nondeterministically choose m ∈ candidates
         in the agenda (list of current activities), create
            an execution stack for τ with method m
   for each stack σ ∈ agenda do
      Progress(σ)
      if σ = ∅ then remove σ from agenda
```

Figure 4: Simplified version of RAE

Rather than requiring the designer to do this, formal tools are needed that will be analogous to the ones for program verification.

- Research is needed on how to acquire operational models of actions and methods. Some promising approaches include combining high level specifications with learning techniques (reinforcement and/or learning from demonstration), and partial programming techniques (Andre and Russell 2002; Simpkins et al. 2008).

Planning with Refinement Methods

For our initial version of a refinement-method planning algorithm, the basic idea is to simulate the effects of RAE's possible choices, to see which ones are likely to be best. Such a *refinement-simulation planner* will have several similarities to the SHOP algorithm (Nau et al. 1999), but also some significant differences. The biggest one is that in SHOP's methods, the method's body was a list of tasks and actions. In a refinement method, the method's body is a program that *generates* tasks and actions. The planner will need to simulate this program's execution. Each time a task is generated, the planner will need to refine that task; and each time a primitive command is generated, the planner will need to predict (perhaps with a precondition-and-effect model, perhaps in some other way) what the command will do.

Research challenges. All of the challenges discussed earlier also arise in planning. Here are some additional ones.

- **Planning over programs**. In our proposed formalism, the body of a method is a simple program, and our proposed refinement-simulation planner just simulates the execution of the program. There are many cases where such an approach will be inadequate, and where it will be necessary to reason about the programs themselves as a planning activity. Reasoning about programs is very hard computationally, and will be infeasible unless some restrictions are placed on the programs. In some cases, this could be done by restricting the programs to be control automata, as in (Bertoli, Pistore, and Traverso 2010; Bucchiarone et al. 2013). Generalizing this approach, and incorporating similar reasoning into a method-decomposition framework, will be significant research challenges.
- **Depth-first versus layered planning.** Rather than a refinement-planning algorithm that works in a depth-first manner, like SHOP, a different approach is to use a layered algorithm like the one in (Marthi, Russell, and Wolfe 2008). That would have some appealing theoretical properties, but the "angelic nondeterminism" technique used in their work requires a way to produce information (bounds on sets of states, and costs of reaching those states) that seems difficult to achieve in practice. Research is needed on how to overcome this problem.

Blended Acting and Planning

The primary reason for doing planning during acting is to help the actor make informed choices. A preliminary solution to this would be to interleave planning and acting, e.g., by having RAE use a planning algorithm to choose which method to use for a task.

Research challenge: For a more thorough blend of planning and acting, the situation is somewhat different; there will be a need to integrate planning into the RAE algorithm itself. What such a blend will look like, we're not sure. This is still purely a "blue sky" idea.

Furthermore, it is worth noting that several of the research challenges that we discussed earlier are more general than *just* planning or *just* acting, and apply to the entire topic of blended planning and acting. These include, for example, the temporal aspects, learning, and verification.

Conclusion

In (Ghallab, Nau, and Traverso 2014) we described the need for research on the design and development of *actors*, as opposed to *planners*, *executors* or other *enablers* that an actor may use to perform its activities. In the current paper we have focused on a key challenge: the discrepancy between the descriptive action models needed for planning, and the operational action models needed for acting.

To address this challenge, we advocate developing a unified hierarchical representation for tasks and actions, that can be used both for planning and for acting (this view will be more extensively developed in our forthcoming book (Ghallab, Nau, and Traverso 2015)). We have outlined what such a representation might be like, and how one might build algorithms that use it for concurrent acting and planning. We have pointed out many places where research is needed to make such an approach successful.

We hope this paper will encourage readers to carry out research on these topics. We are optimistic that such research will provide a solid foundation for development of highly capable actors for a variety of applications.

Acknowledgments. This work was supported in part by EU FP7 SAPHARI grant ICT-287513, ARO grant W911NF1210471, and ONR grants N000141210430 and N000141310597. The information in this paper does not necessarily reflect the position or policy of the funders.

References

Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI*, 1–7.

Barbier, M.; Gabard, J.-F.; Llareus, J. H.; and Tessier, C. 2006. Implementation and flight testing of an onboard architecture for mission supervision. In *International Unmanned Air Vehicle Systems Conference*.

Beetz, M., and McDermott, D. 1994. Improving robot plans during their execution. In *AIPS*, volume 2.

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM – a cognitive robot abstract machine for everyday manipulation in human environments. In *ICRA*, 1–6.

Bertoli, P.; Pistore, M.; and Traverso, P. 2010. Automated composition of web services via planning in asynchronous domains. *Artif. Intell.* 174(3-4):316–361.

Bohren, J.; Rusu, R. B.; Jones, E. G.; Marder-Eppstein, E.; Pantofaru, C.; Wise, M.; Mösenlechner, L.; Meeussen, W.; and Holzer, S. 2011. Towards autonomous robotic butlers: Lessons learned with the pr2. In *ICRA*, 5568–5575.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *JAAMAS* 19(3):297–331.

Bucchiarone, A.; Marconi, A.; Pistore, M.; Traverso, P.; Bertoli, P.; and Kazhamiakin, R. 2013. Domain objects for continuous context-aware adaptation of service-based systems. In *ICWS*, 571–578.

Conrad, P.; Shah, J.; and Williams, B. C. 2009. Flexible execution of plans with choice. In *ICAPS*.

Dean, T. L., and Wellman, M. 1991. *Planning and Control*. Morgan Kaufmann.

desJardins, M.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Mag.* 20(4):13–22.

Despouys, O., and Ingrand, F. 1999. Propice-Plan: Toward a unified framework for planning and execution. In *European Workshop on Planning*.

Effinger, R.; Williams, B.; and Hofmann, A. 2010. Dynamic execution of temporally and spatially flexible reactive programs. In *AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 1–8.

Fikes, R. E. 1971. Monitored Execution of Robot Plans Produced by STRIPS. In *IFIP Congress*.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *AAAI Conference*. Seattle, WA.

Garcia, C. E.; Prett, D. M.; and Morari, M. 1989. Model predictive control: theory and practice–a survey. *Automatica* 25(3):335–348.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actor's view of automated planning and acting: A position paper. *Artif. Intell.* 208:1–17.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2015. *Automated Planning and Acting*. Cambridge University Press. To appear.

Hähnel, D.; Burgard, W.; and Lakemeyer, G. 1998. GOLEX—bridging the gap between logic (GOLOG) and a real robot. In *KI*, 165–176.

Ingham, M. D.; Ragno, R. J.; and Williams, B. C. 2001. A reactive model-based programming language for robotic space explorers. In *i-SAIRAS*.

Ingrand, F., and Ghallab, M. 2015. Deliberation for Autonomous Robots: A Survey. *Artif. Intell.* (To appear).

Ingrand, F.; Chatilla, R.; Alami, R.; and Robert, F. 1996. PRS: A high level supervision and control language for autonomous mobile robots. In *ICRA*, 43–49.

Kortenkamp, D., and Simmons, R. 2007. Robotics systems architectures and programming. In Khatib, and Siciliano., eds., *Handbook of Robotics*. Springer.

Löhr, J.; Eyerich, P.; Keller, T.; and Nebel, B. 2012. A planning based framework for controlling hybrid systems. In *ICAPS*.

Marthi, B.; Russell, S.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, 222–231.

Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In Dean, T., ed., *IJCAI-99*, 968–973. Morgan Kaufmann.

Pappachan, P. M., and Durfee, E. H. 2000. Interleaved plan coordination and execution in dynamic multi-agent domains. In *ICMAS*, 425–426.

Pollack, M. E., and Horty, J. F. 1999. There's more to life than making plans: Plan management in dynamic, multiagent environments. *AI Mag.* 20(4):1–14.

Rosenschein, S., and Kaelbling, L. P. 1986. Integrating planning and reactive control.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *IROS*, 1931–1937 vol.3.

Simmons, R. 1992. Concurrent planning and execution for autonomous robots. *Control Systems, IEEE* 12(1):46–50.

Simpkins, C.; Bhat, S.; Isbell, Jr., C.; and Mateas, M. 2008. Towards adaptive programming: integrating reinforcement learning into a programming language. In *Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA '08, 603–614. ACM.

Verma, V.; Estlin, T.; Jónsson, A. K.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In *i-SAIRAS*.

Wang, F. Y.; Kyriakopoulos, K. J.; Tsolkas, A.; and Saridis, G. N. 1991. A Petri-net coordination model for an intelligent mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics* 21(4):777–789.

Williams, B. C., and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*.

Ziparo, V. A.; Iocchi, L.; Lima, P. U.; Nardi, D.; and Palamara, P. F. 2011. Petri net plans. *AAMAS* 23(3):344–383.