Advances in Cognitive Systems X (20XX) 1-6 Submitted X/20XX; published X/20XX

# Refinement Planning and Acting

**Dana Nau** NAU@CS.UMD.EDU
Dept. 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 several publications over the past few years, we argued that the automated planning research literature has underestimated the importance and difficulty of deliberative acting. We have outlined an algorithm, RAE, for deliberative acting. We now give give an overview of SeRPE, a planning algorithm designed to be integrated with RAE. Additional details about both RAE and SeRPE are in Chapter 3 of our new book, *Automated Planning and Acting*.

## 1. Introduction

In (Ghallab et al., 2014; Traverso et al., 2015; Nau et al., 2015), we argued that the automated planning research literature has underestimated the importance and difficulty of *deliberative acting*, and we called for more research on the problems that emerge when integrating acting with planning. Our main points were as follows:

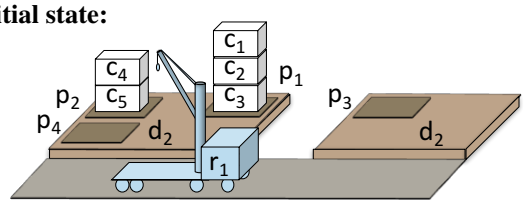
- Acting is more than just executing actions. Actors often are collections of hierarchically organized modules, in which a component receives tasks from the component above it, and must decide how to perform those tasks. This may involve refining the task into lower-level steps, issuing subtasks to lower-level components, issuing commands to the execution platform.
- To carry out this deliberative process, the actor uses *operational* models telling it *how* to perform various tasks under various circumstances. These differ from the *descriptive* models used by planners, which describe *what* the actor's actions will do, not how to do them.
- So that the actor can respond quickly to unpredicted events, planning needs to be both online and continual. 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.

As initial steps toward developing an actor with the above capabilities, Nau et al. (2015) outlined a formalism for *refinement methods* (hierarchically organized operational models describing ways that an actor can perform a given task), and gave partial pseudocode for RAE (Refinement Acting

**Objects:**

robot $\mathsf{r}_1$, containers $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_4, \mathsf{c}_5$,
loading docks $\mathsf{d}_1, \mathsf{d}_2$, piles $\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4$.

**Initial state:**




**Commands:**

$\mathsf{go}(r, d, d')$: robot $r$ moves from dock $d$ to $d'$
$\mathsf{load}(r, c, c', p, d)$: robot $r$ takes container $c$
$\mathsf{unload}(r, c, c', p, d)$: if pile $p$ has $\leq 4$ containers, then $r$ puts $c$ onto the top of $p$, otherwise the command fails

**Tasks:**

$\mathsf{put\text{-}in\text{-}pile}(c, q)$: put container $c$ in pile $q$
$\mathsf{uncover}(c)$: ensure no containers are on $c$

**Refinement methods:**

$\mathsf{m\text{-}put\text{-}in\text{-}pile}(r, c, p, q)$
task: $\mathsf{put\text{-}in\text{-}pile}(c, q)$
pre: $\mathsf{pile}(c) = p \wedge \mathsf{cargo}(r) = \mathsf{nil}$
body: if $\mathsf{pile}(c) \neq q$ then
$\mathsf{uncover}(c)$
if $\mathsf{loc}(r) \neq \mathsf{loc}(p)$ then $\mathsf{go}(r, \mathsf{loc}(p))$
$\mathsf{load}(r, c, \mathsf{pos}(c), p, \mathsf{loc}(p))$
if $\mathsf{loc}(r) \neq \mathsf{loc}(q)$ then $\mathsf{go}(r, \mathsf{loc}(q))$
$\mathsf{unload}(r, c, \mathsf{top}(q), q, \mathsf{loc}(q))$

$\mathsf{m\text{-}uncover}(r, c, p, q)$
task: $\mathsf{uncover}(c)$
pre: $\mathsf{pile}(c) = p \wedge p \neq q \wedge \mathsf{cargo}(r) = \mathsf{nil}$
body: while $\mathsf{top}(p) \neq c$ do
if $\mathsf{loc}(r) \neq \mathsf{loc}(p)$ then $\mathsf{go}(r, \mathsf{loc}(p))$
$c' \leftarrow \mathsf{top}(p)$
$\mathsf{load}(r, c', \mathsf{pos}(c'), p, \mathsf{loc}(p))$
if $\mathsf{loc}(r) \neq \mathsf{loc}(q)$ then $\mathsf{go}(r, \mathsf{loc}(q))$
$\mathsf{unload}(r, c', \mathsf{top}(q), q, \mathsf{loc}(q))$

*Figure 1.* A simple example of an actor, its capabilities, and its environment.

Engine), which uses refinement methods for deliberative acting. We now describe a planner, SeRPE, which simulates RAE's operation in order to do such a lookahead. More details about RAE, SeRPE, and the refinement method formalism are in Chapter 3 of our new book, Ghallab et al. (2016).

## 2. Motivating Example

Figure 1 gives a simple example. There are a robot vehicle, some loading docks, and containers that can be stacked in piles. The robot can send *commands* to its execution platform, and it has *refinement methods* telling it how to use those commands to perform two kinds of *tasks*: uncovering a container (i.e., ensuring no other containers are on it), and putting the container on top of a pile.

Consider the task $\mathsf{put\text{-}in\text{-}pile}(\mathsf{c}_1, \mathsf{p}_3)$. For this task, the relevant method instance is $\mathsf{m\text{-}put\text{-}in\text{-}pile}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{p}_1, \mathsf{p}_3)$, whose body includes the task $\mathsf{uncover}(\mathsf{c}_3)$. For this task there are several relevant method instances. One of them is $\mathsf{m\text{-}uncover}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{p}_1, \mathsf{p}_2)$, which will produce the first refinement tree in Figure 2. In that refinement tree, RAE attempts to move the containers to pile $\mathsf{p}_2$, but this fails because it would make $\mathsf{p}_2$ too high. Although it is possible to recover from this error, the error can be avoided completely if RAE instead chooses a different method instance, $\mathsf{m\text{-}uncover}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{p}_1, \mathsf{p}_4)$. This generates the second refinement tree, in which RAE successfully moves the containers to pile $\mathsf{p}_4$.

RAE makes choices reactively, without planning ahead to see what problems might occur. If RAE could plan ahead, it would be able to see the problem with $\mathsf{m\text{-}uncover}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{p}_1, \mathsf{p}_2)$ and choose $\mathsf{m\text{-}uncover}(\mathsf{r}_1, \mathsf{c}_1, \mathsf{p}_1, \mathsf{p}_4)$ instead. This motivates the need for a planner.

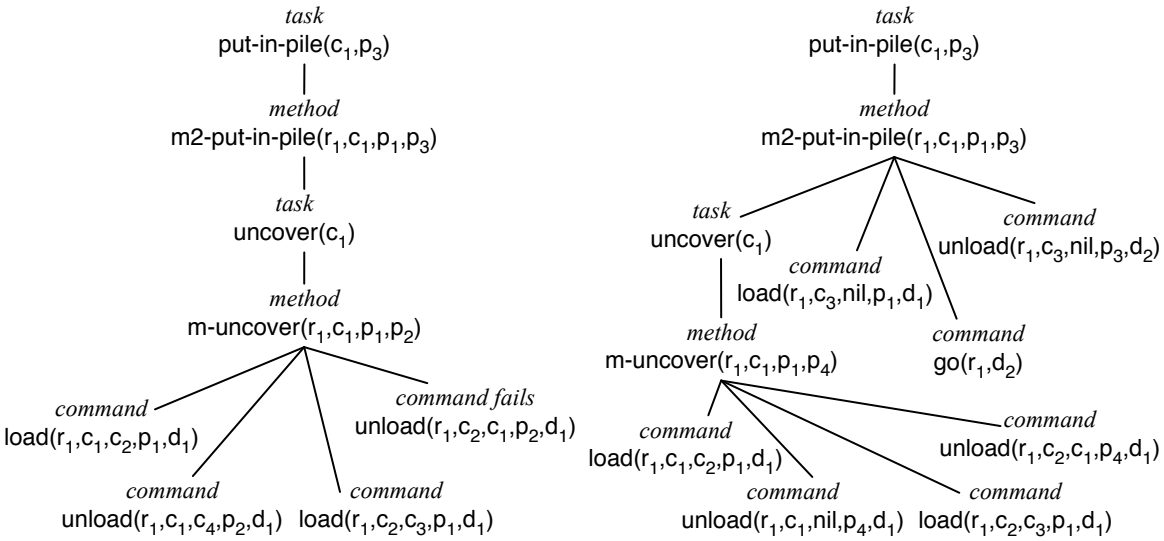



*Figure 2.* Two refinement trees for the task put-in-pile$(c_1, p_3)$. The first one fails, the second one succeeds.

SeRPE$(\mathcal{M}, \mathcal{A}, s, \tau)$
  *Candidates* $\leftarrow$ Instances$(\mathcal{M}, \tau, s)$
  if *Candidates* $= \varnothing$ then return failure
  nondeterministically choose $m \in$ *Candidates*
  return Progress-to-finish$(\mathcal{M}, \mathcal{A}, s, \tau, m)$

Progress-to-finish$(\mathcal{M}, \mathcal{A}, s, \tau, m)$
  $\pi \leftarrow \langle\rangle$  *// current plan*
  loop
    $i \leftarrow$ nextstep$(m, i)$  *// instruction pointer*
    case type$(m[i])$
      assignment:
        update $s$ according to $m[i]$
      command:
        $a \leftarrow$ descriptive model of $m[i]$
        if $s \models$ pre$(a)$ then $s \leftarrow \gamma(s, a)$; $\pi \leftarrow \pi.a$
        else return failure
      task:
        $\pi' \leftarrow$ SeRPE$(\mathcal{M}, \mathcal{A}, s, m[i])$
        if $\pi' =$ failure then return failure
        $s \leftarrow \gamma(s, \pi')$; $\pi \leftarrow \pi.\pi'$

**Action models:**

go$(r, d, d')$
  pre: loc$(r) = d$
  eff: loc$(r) \leftarrow d'$

load$(r, c, c', p, d)$
  pre: at$(p, d)$, cargo$(r) =$ nil, loc$(r) = d$,
    pos$(c) = c'$, top$(p) = c$
  eff: cargo$(r) = c$, pile$(c) \leftarrow$ nil, pos$(c) \leftarrow r$,
    top$(p) \leftarrow c'$, height$(p) \leftarrow$ height$(p) - 1$

unload$(r, c, c', p, d)$
  pre: at$(p, d)$, pos$(c) = r$, loc$(r) = d$,
    top$(p) = c'$, height$(p) \leq 4$
  eff: cargo$(r) \leftarrow$ nil, pile$(c) \leftarrow p$, pos$(c) \leftarrow c'$,
    top$(p) \leftarrow c$, height$(p) \leftarrow$ height$(p) + 1$

*Figure 3.* A slightly simplified version of SeRPE, and action models for the commands in Figure 1.

## 3. SeRPE (Sequential Refinement Planning Engine)

A key challenge in integrating acting with planning is the discrepancy between the operational models needed for acting and the descriptive models needed for planning. One way to address this

challenge is to observe that although RAE's refinement methods were designed as operational models, they still can be used for planning. We have written a planning algorithm, SeRPE (Sequential Refinement Planning Engine), to do that. SeRPE still needs to use descriptive action models in place of RAE's commands to the execution platform—but since it can use RAE's refinement methods, this makes it much easier to maintain consistency between RAE and SeRPE.

SeRPE (see Figure 3) uses RAE's refinement methods to *simulate* RAE's possible execution paths. At each point where RAE would send a command to its execution platform, SeRPE predicts the command's outcome using a descriptive action model. For example, SeRPE can generate both of the refinement trees in Figure 2, and predict that the first one will fail.

SeRPE's task refinement process has some similarity to HTN planners such as SHOP (Nau et al., 1999), but with important difference: unlike HTN planning, a refinement method's body isn't a fixed sequence of subtasks. Instead, it is a computer program (e.g., the *while* loop in m-uncover in Figure 1), and executing this computer program will *generate* a sequence of subtasks. If the body of a method contains a task such as uncover($c_1$), for which there are several relevant method instances, then the body of the method has several possible execution paths.

This presents a key implementation issue: for SeRPE to explore the different possible choices, it cannot just execute the body of a method as ordinary computer code. Instead, it needs a way to generate multiple execution paths for the body of a method. Some of our students are doing an implementation of RAE and SeRPE, and building such a facility into SeRPE.

For future work, one limitation of SeRPE is that its action models must be deterministic, hence they cannot adequately simulate commands that have multiple possible outcomes. For example, it is easy (see Ghallab et al. (2016)) to write refinement methods for RAE to search for an object, but SeRPE cannot use these methods to predict where the object will be found. To overcome this limitation, in our future research we want to generalize SeRPE to use nondeterministic action models.

## Acknowledgements

This work was supported in part by ONR grant N000141310597. We also acknowledge the support of our respective institutions: University of Maryland, LAAS/CNRS, and FBK ICT IRST.

## References

Ghallab, M., Nau, D., & Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, *208*, 1–17.

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge U. Press.

Nau, D., Ghallab, M., & Traverso, P. (2015). Blended planning and acting: Preliminary approach, research challenges. *Proc. AAAI*.

Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proc. IJCAI* (pp. 968–973).

Traverso, P., Ghallab, M., & Nau, D. S. (2015). An IPC track on deliberative acting: Moving the competition ahead towards more relevant scientific challenges. *The International Planning Competition (WIPC-15)* (p. 29).