Hierarchical Refinement as a Generalization of HTN Planning

Dana Nau
University of Maryland

work performed with

- Sunandita Patra University of Maryland
- Malik Ghallab LAAS/CNRS, University of Toulouse
- Paolo Traverso FBK ICT IRST, Trento, Italy
- James Mason University of Maryland

Harbor Management Tasks

- Multiple levels of abstraction
  - Physical/managerial organization of harbor

- Upper levels:
  - Abstract tasks, can be planned in advance

- Lower levels:
  - Multiple agents
  - Partial observability
  - Dynamic change

- Continual online planning
  - Abstract and partial until more detail needed
Acting and Planning

Acting
- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
  - Adapt to context, react to events

Planning
- *Prediction* + *search*
- Search over predicted states, possible organizations of tasks and actions
- Use *descriptive models* to predict *what*
- Planning in service of acting
  - Actor asks planner for advice
  - Planner runs online
    - e.g., receding horizon
Opening a Door

- Details depends on what kind of door
  - Might not be known until acting time
What kind of door?

- Sliding or hinged

- Identify type of door
- Move close to knob
- Grasp knob
- Turn knob
- Maintain
- Pull
- Monitor
- Move back
- Ungrasp

- Open door
- Close door
- Pull
- Monitor
- Move to door
- Fetch
- Navigate to room2
- Navigate to room1
- Open door

What kind of door?

Ø

Sliding or hinged
What kind of door?

- Sliding or hinged
- Hinge on left or right
What kind of door?

- Sliding or hinged
- Hinge on left or right
- Open toward or away
What kind of door?

- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever,
What kind of door?

- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever, push bar, push plate,

identify type of door
move close to knob
grasp knob
turn knob
monitor
monitor
move back
ungrasp

open door

get out

respond to user requests

bring o7 to room2
go to hallway
deliver o7

move to door
fetch o7
navigate to room2
navigate to room1
open door
What kind of door?

- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever, push bar, push plate, pull handle, thumb latch,
What kind of door?

- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever, push bar, push plate, pull handle, thumb latch, something else
**Refinement Acting**

- **Task**: 
  - activity for the actor to perform
  - For each task, one or more **refinement methods**
  - Operational models telling how to perform the task

```
method-name(arg_1, ..., arg_k)
task: task-identifier
pre: test
body: computer program
      that may include tasks and commands
```

“primitive” functions that the actor can send to its execution platform
Refinement Acting

- **Task:**
  - activity for the actor to perform
  - For each task, one or more *refinement methods*
    - Operational models telling how to perform the task

  method-name\((arg_1, \ldots, arg_k)\)
  - task: *task-identifier*
  - pre: *test*
  - body: *computer program that may include tasks and commands*

- **Differences from HTN methods**
  - Actor uses them reactively
  - Body is a computer program that invokes tasks, commands
  - Commands interact with external world
  - Outcomes not known in advance
  - Current state obtained using sensors, represented using state variables

  "primitive" functions that the actor can send to its execution platform
Opening a Door

- What kind:
  - Hinged on left, opens toward us, knob

m1-unlatch($r,d,l,o$)

- task: **unlatch**($r,d$)
- pre: $\text{loc}(r,l) \land \text{toward-side}(l,d) \land$ side($d$,left) $\land$ type($d$,rotate) $\land$ handle($d$,o)
- body: grasp($r,o$)
  - turn($r,o,alpha1$)
  - pull($r,val1$)
  - if door-status($d$) = cracked then **ungrasp**($r,o$)
  - else fail

m1-throw-wide($r,d,l,o$)

- task: **throw-wide**($r,d$)
- pre: $\text{loc}(r,l) \land \text{toward-side}(l,d) \land$ side($d$,left) $\land$ type($d$,rotate) $\land$ handle($d$,o) $\land$ door-status($d$) = cracked
- body: grasp($r,o$)
  - pull($r,val1$)
  - move-by($r,val2$)
RAE (Reactive Acting Engine)

- Uses refinement methods to accomplish tasks
  - Based on OpenPRS robot control architecture

- I’ll give a summary

- For details:
  - Final manuscript and lecture slides freely downloadable [here](#)
RAE (Reactive Acting Engine)

Agenda: \(\{\text{stack } \sigma_1, \ldots, \text{stack } \sigma_n\}\)
Like program execution stacks

\[
\text{stack} = \begin{pmatrix}
\text{(sub-subtask, method, \ldots)} \\
\text{(subtask, method, \ldots)} \\
\text{(task, method, \ldots)}
\end{pmatrix}
\]

loop:
for every new external task or event \(\tau\)

*Candidates* = \{applicable method instances\}
choose \(m \in \text{Candidates}\)
create refinement stack \(\sigma\), \textit{initially just } \langle \tau, m \rangle
add \(\sigma\) to Agenda
for each stack \(\sigma\) in Agenda
Progress(\(\sigma\))
if \(\sigma\) is finished, remove it from Agenda

Progress(\(\sigma\)):
\[
(\tau, m, \ldots) \leftarrow \text{top}(\sigma)
\]

is \(m\)'s current step a command?

*executing*

command status?

*finished*

more steps in \(m\)?

*failed*

retry \(\tau\) using an untried candidate

*success*

return

*no*

pop(\(\sigma\))

\(a \leftarrow \text{next step of } m\)

*a's type?*

*assignment*

command

update state

candidates for \(a\)?

*yes*

choose a candidate \(m'\)
push \((a, m', \ldots)\) onto \(\sigma\)

*no*

send \(a\) to the execution platform

retry \(\tau\) using an untried candidate
**RAE (Reactive Acting Engine)**

**Agenda:** \{stack $\sigma_1$, ..., stack $\sigma_n$\}

Like program execution stacks

<table>
<thead>
<tr>
<th>stack $\sigma$ =</th>
</tr>
</thead>
<tbody>
<tr>
<td>(sub-subtask, method, ...)</td>
</tr>
<tr>
<td>(subtask, method, ...)</td>
</tr>
<tr>
<td>(task, method, ...)</td>
</tr>
</tbody>
</table>

**loop:**

for every new external task or event $\tau$

- **Candidates** = \{applicable method instances\}

choose $m \in$ **Candidates**

create refinement stack $\sigma$, initially just $\langle \tau, m \rangle$

add $\sigma$ to **Agenda**

for each stack $\sigma$ in **Agenda**

* Progress($\sigma$) *

if $\sigma$ is finished, remove it from **Agenda**

---

**Progress($\sigma$):**

$(\tau, m, \ldots) \leftarrow \text{top}(\sigma)$

no

- is $m$’s current step a command?

  yes

  command status?

  finished

  more steps in $m$?

  yes

  $a \leftarrow \text{next step of } m$

  command $a$’s type?

  for each stack $\sigma$ in **Agenda**

  Progress($\sigma$)

  if $\sigma$ is finished, remove it from **Agenda**

- no

  pop($\sigma$)

- no

  retry $\tau$ using an untried candidate

---

- Get advice from a planner

---

Nau: ICAPS hierarchical planning workshop, 2019
How to Do the Planning?

- In the book: SeRPE planner
  - Extends the SHOP algorithm to reason about refinement methods
  - Executes code in the method’s body, but
    - Descriptive models of commands
    - Classical actions, abstract state
  - To backtrack from a method \( m \):
    - Revert to state when \( m \) was chosen

**m-opendoor\((r,d,l,h)\)**
- **task:** `opendoor\((r,d)\)`
- **pre:** \( \text{loc}(r) = l \land \text{road}(l,d) \land \text{handle}(d,h) \)
- **body:**
  - while \( \neg \text{reachable}(r,h) \) do
    - `move-close\((r,h)\)`
    - `monitor-status\((r,d)\)`
  - if `door-status\((d)\) = closed` then
    - `unlatch\((r,d)\)`
    - `throw-wide\((r,d)\)`
    - `end-monitor-status\((r,d)\)`
How to Do the Planning?

- In the book: SeRPE planner
  - Extends the SHOP algorithm to reason about refinement methods
  - Executes code in the method’s body, but
    - Descriptive models of commands
    - Classical actions, abstract state
  - To backtrack from a method $m$:
    - Revert to state when $m$ was chosen

Classical actions don’t represent
- Nondeterministic outcomes, partial observability, exogenous events, durations, predicted future events, overlapping actions and events, ...
- Sometimes OK, but often not
Acting and Planning

**Acting**
- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
  - Adapt to context, react to events

**Planning**
- *Prediction* + *search*
- Search over predicted states, possible organizations of tasks and actions
- Use *descriptive models* to predict *what*

**Consistency?**
Acting and Planning

Acting
- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
- Adapt to context, react to events

Planning
- *Prediction + search*
- Search over predicted states, possible organizations of tasks and actions
- Use *descriptive models* to predict *what*

Simulated execution of actor’s operational models
Why should we think this will work?

- Why does AI planning use descriptive models?
  1. AI planners search a huge search space, need fast predictions
  2. Effort required to create detailed operational models
     - Especially if you aren’t a domain expert

- Problems aren’t as bad as they used to be
  1. Like HTN methods, the operational models focus the search
     Computers have gotten more powerful
     - Compute detailed simulations more quickly
       - e.g., fast physics-based simulations
  2. Real-world actors will already include control software
     - May be able to use it as refinement methods
**RAEplan**

- Simulate RAE on a single task

RAEplan(task \( \tau \)):

*Candidates* = \{applicable method instances\}

nondeterministically choose \( m \in \text{Candidates} \)

create refinement stack \( \sigma \) for \( \tau \) and \( m \)

loop while Progress(\( \sigma \)) \( \neq \) failure

if \( \sigma \) is completed then return \( m \)

return *failure*

**Progress(\( \sigma \))**: 

\((\tau, m, \ldots) \leftarrow \text{top}(\sigma)\)

- if \( m \)'s current step a command?
- if \( m \)'s command status?
- if \( m \)'s more steps in \( m \)?
- if \( m \)'s finished
- if \( m \)'s no more steps in \( m \)?
- if \( m \)'s no candidates for \( m \)?

- return

return *failure*

- return

- \( \text{top}(\sigma) \)

- \( \text{pop}(\sigma) \)

- \( a \leftarrow \text{next step of } m \)

- assignment

- command

- task

- candidates for \( a \)?

- choose a candidate \( m' \)

push \((a, m', \ldots)\) onto \( \sigma \)

- either simulate or use descriptive model

- update state

- yes

- no

- return *failure*
RAEplan

- Simulate RAE on a single task

**RAEplan(task \( \tau \)):**

*Candidates* = \{applicable method instances\}

nondeterministically choose \( m \in \text{Candidates} \)

create refinement stack \( \sigma \) for \( \tau \) and \( m \)

loop while \( \text{Progress}(\sigma) \neq \text{failure} \)

if \( \sigma \) is completed then return \( m \)

return \text{failure}

---

**Progress(\( \sigma \)):**

(\( \tau, m, \ldots \)) \leftarrow \text{top}(\sigma)

- is \( m \)'s current step a command?

  - yes

    - \text{executing command status?}

      - failed

        - return failure

      - finished

        - more steps in \( m \)?

          - no

            - \text{return}

          - yes

            - pop(\( \sigma \))

    - no

      - assignment

        - \( a \)’s type?

          - command

            - candidates for \( a \)?

              - yes

                - update state

              - no

                - return failure

              - either simulate or use descriptive model

            - task

              - choose a candidate \( m' \)

                - push \((a, m', \ldots)\) onto \( \sigma \)

        - return failure

      - return failure

---

Nau: ICAPS hierarchical planning workshop, 2019
RAEplan

- Simulate RAE on a single task

RAEplan(task τ):

Candidates = {applicable method instances}
non-deterministically choose m ∈ Candidates
create refinement stack σ for τ and m
loop while Progress(σ) ≠ failure
  if σ is completed then return m
return failure

Idea 1: backtracking

- Backtrack over method’s body
- Was easy in SHOP
  - backtrack over the task list
- In RAEplan, need to backtrack over code
- A group of students worked on this for several months, didn’t get it to work

Progress(σ): (τ, m, …) ← top(σ)

no

is m’s current step a command?

yes

executing

command status?

failed

return

finished

more steps in m?

no

return failure

yes

pop(σ)

a ← next step of m

candidates for a?

no

return failure

yes

choose a candidate m’

push (a, m’, …) onto σ

either simulate or use descriptive model

class assignment

class command

class task

class update state

class either simulate or use descriptive model

Nau: ICAPS hierarchical planning workshop, 2019
**RAEplan**

- Simulate RAE on a single task

**RAEplan**(task \( \tau \)):
  - Candidates = \{applicable method instances\}
  - nondeterministically choose \( m \in \text{Candidates} \)
  - create refinement stack \( \sigma \) for \( \tau \) and \( m \)
  - loop while Progress(\( \sigma \)) \( \neq \) failure
  - if \( \sigma \) is completed then return \( m \)
  - return failure

**Idea 2:** multithreading

- Too many parallel processes

**Progress(\( \sigma \))**:
  - (\( \tau, m, \ldots \)) \( \leftarrow \) top(\( \sigma \))
  - is \( m \)'s current step a command?
    - no
      - return
    - yes
      - command status?
        - failed
          - return failure
        - executing
        - finished
          - more steps in \( m \)?
            - no
              - pop(\( \sigma \))
            - yes
              - \( a \leftarrow \) next step of \( m \)
        - a's type?
          - assignment
            - update state
            - yes
              - candidates for \( a \)?
                - yes
                  - either simulate or use descriptive model
                - no
                  - return failure
            - no
          - command
            - task
              - candidates for \( a \)?
                - yes
                  - either simulate or use descriptive model
                - no
                  - return failure
            - no
          - return failure

**RAEplan** = Too many parallel processes

Nau: ICAPS hierarchical planning workshop, 2019
**RAEplan**

- Simulate RAE on a single task

**RAEplan(task τ):**

1. *Candidates* = {applicable method instances}
2. Nondeterministically choose *m* ∈ *Candidates*
3. Create refinement stack *σ* for *τ* and *m*
4. Loop while *Progress(σ) ≠ failure*
   - If *σ* is completed then return *m*
   - Return *failure*

---

**Idea 3:** Monte Carlo rollouts

- Multiple runs
  - Random outcomes in each run
- Return the method *m* that gives the highest expected utility

---

Summary of Experimental Results

<table>
<thead>
<tr>
<th>Domain</th>
<th>Dynamic events</th>
<th>Dead ends</th>
<th>Sensing</th>
<th>Robot collaboration</th>
<th>Concurrent tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>EE</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SD</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IP</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Four different domains, different combinations of characteristics
- Evaluation criteria:
  - Efficiency, successes vs failures, how many retries
- Result: planning helps
  - RAE operated better with RAEplan than without
  - RAE operated better with more planning than with less planning

Patra, Traverso, Ghallab, and Nau. Acting and planning using operational models. *AAAI*, 2019
Summary

- Continual online planning, in service of acting
- Descriptive vs. operational models
- Refinement methods
  - Generalization of HTN methods
    - State variables
    - Body is a computer program that includes tasks, commands
    - Commands interact with environment
    - Outcomes not known in advance
  - Used in two ways
    - Reactively in RAE
    - Predictively in RAEplan
- Experimental results with RAE, RAEplan
  - More planning → better acting
Limitations and Future Work

- How to get the operational models?
  - Earlier, I said, “Real-world actors may already include operational models”
    - OK if the actor is RAE, OpenPRS, …
    - Otherwise, might not be in a form we can use
  - Currently, only alternative is to write them ourselves
    - Develop learning algorithms to do this?

- Experiments were done in “toy domains”
  - Want to test the approach in real planning problems

- Ongoing project with US Naval Research Laboratory
  - Use RAE, RAEplan for recovery from attacks on software-defined networks
  - They’re writing the refinement methods
  - We plan to modify RAE, RAEplan to meet their needs
Links

  - Final manuscript, lecture slides

- Patra, Traverso, Ghallab, and Nau. *Acting and planning using operational models*. *AAAI*, 2019

- **RAE and RAeplan source code**
  - 3-clause BSD license
  - Caveat: software still under development