Chapter 3
Deliberation with Refinement Methods

Section 3.1: Operational Models
Section 3.2: A Refinement Acting Engine

Dana S. Nau
University of Maryland

http://www.laas.fr/planning
Motivation

- Deliberation is hierarchically organized
  - At high levels, the actions are abstract
  - At lower levels, more detail

- Refine abstract actions into ways of carrying out those actions
  - How?
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?

![Diagram of different types of doors and actions for opening them]
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, …
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, …
Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, something else?
Outline

3.1 Representation
   a. State variables, commands, refinement methods
   b. Example

3.2 Acting
   a. Rae (Refinement Acting Engine)
   b. Example
   c. Extensions

3.3 Planning
   a. Motivation and basic ideas
   b. Deterministic action models
   c. SeRPE (Sequential Refinement Planning Engine)

3.4 Using Planning in Acting
   a. Techniques
   b. Caveats
State-Variable Representation

Quick review:

- **Objects:** $\text{Robots} = \{\text{rbt}\}$, $\text{Containers} = \{\text{c1}, \text{c2}, \text{c3}, \ldots\}$,
  $\text{Locations} = \{\text{loc0}, \text{loc1}, \text{loc2}, \ldots\}$

- **State variables**: syntactic terms to which we can assign values
  - $\text{loc}(r) \in \text{Locations}$
  - $\text{load}(r) \in \text{Containers} \cup \{\text{nil}\}$
  - $\text{pos}(c) \in \text{Locations} \cup \text{Robots} \cup \{\text{unknown}\}$
  - $\text{view}(r,l) \in \{\text{T}, \text{F}\}$ – whether robot $r$ has looked at location $l$
    - $r$ can only see what’s at its current location

- **State**: assign a value to each state variable
  - $\{\text{loc}(\text{rbt}) = \text{loc0}, \text{pos}(\text{c1}) = \text{loc2}, \text{pos}(\text{c3}) = \text{loc4}, \text{pos}(\text{c2}) = \text{unknown}, \ldots\}$

Details: *Automated Planning and Acting*, Sections 2.1 and 3.1.1
State-Variable Representation

Extensions:

- **Range(x)**
  - can be finite, infinite, continuous, discontinuous, vectors, matrices, other data structures

- **Assignment statement** \( x \leftarrow expr \)
  - expression that returns a ground value in Range(x)
  - and has no side-effects on the current state

- **Tests (e.g., preconditions)**
  - **Simple**: \( x = v, \ x \neq v, \ x > v, \ x < v \)
  - **Compound**: conjunction, disjunction, or negation of simple tests
**Commands**

*Command*: primitive function that the execution platform can perform

- $\text{take}(r,o,l)$: robot $r$ takes object $o$ at location $l$
- $\text{put}(r,o,l)$: $r$ puts $o$ at location $l$
- $\text{perceive}(r,l)$: robot $r$ perceives what objects are at $l$
  - $r$ can only perceive what’s at its current location
- ...
Tasks and Methods

- **Task**: an activity for the actor to perform
- For each task, a set of refinement methods*
  - **Operational** models:
    - tell *how* to perform the task
    - don’t predict *what* it will do

```
method-name(arg_1, ..., arg_k)
task: task-identifier
pre: test
body: a program
```

- assignment statements
- control constructs: if-then-else, while, … .
- tasks (can extend to include events, goals)
- commands to the execution platform
Opening a Door

- What kind:
  - Hinged on left, opens toward us, lever handle

```
m-open-door(r,d,l,h)
  task: open-door(r,d)
  pre: loc(r) = l \land \text{adjacent}(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) = \text{closed} then
    unlatch(r,d)
  else
    fail

m1-unlatch(r,d,l,o)
  task: unlatch(r,d)
  pre: loc(r,l) \land \text{toward-side}(l,d) \land
    \text{side}(d,\text{left}) \land \text{type}(d,\text{rotate}) \land \text{handle}(d,o)
  body: grasp(r,o)
    turn(r,o,\text{alpha1})
    pull(r,\text{val1})
    if door-status(d) = \text{cracked} then ungrasp(r,o)
    else fail

m1-throw-wide(r,d,l,o)
  task: throw-wide(r,d)
  pre: loc(r,l) \land \text{toward-side}(l,d) \land
    \text{side}(d,\text{left}) \land \text{type}(d,\text{rotate}) \land
    \text{handle}(d,o) \land \text{door-status}(d) = \text{cracked}
  body: grasp(r,o)
    pull(r,\text{val1})
    move-by(r,\text{val2})
```
Outline

3.1 Representation
   a. State variables, commands, refinement methods
   b. Example

3.2 Acting
   a. Rae (Refinement Acting Engine)
   b. Example
   c. Extensions

3.3 Planning
   a. Motivation and basic ideas
   b. Deterministic action models
   c. SeRPE (Sequential Refinement Planning Engine)

3.4 Using Planning in Acting
   a. Techniques
   b. Caveats
Rae (Refinement Acting Engine)

- Based on OpenPRS
  - Programming language, open-source robotics software
  - Deployed in many applications

- Input: external tasks, events, current state
- Output: commands to execution platform

- Perform multiple tasks/events in parallel
  - Purely reactive, no lookahead

- For each task/event, a refinement stack
  - current path in Rae’s search tree for the task/event

- Agenda = {all current refinement stacks}
Basic idea: 

\[ Agenda = \{\text{current refinement stacks}\} \]

loop:
  if new external tasks/events then add them to Agenda
  for each stack in Agenda
    Progress it, and remove it if it’s finished

\[ \text{Rae}(\mathcal{M}) \]

\[ Agenda \leftarrow \emptyset \]

loop 
  until the input stream of external tasks and events is empty do 
    read \( \tau \) in the input stream 
    \[ \text{Candidates} \leftarrow \text{Instances}(\mathcal{M}, \tau, \xi) \]
    if \( \text{Candidates} = \emptyset \) then output(“failed to address” \( \tau \)) 
    else do 
      arbitrarily choose \( m \in \text{Candidates} \)
      \[ Agenda \leftarrow Agenda \cup \{((\tau, m, \text{nil}, \emptyset))\} \]
      for each \( stack \in Agenda \) do 
        Progress(\( stack \))
        if \( stack = \emptyset \) then \( Agenda \leftarrow Agenda \setminus \{\text{stack}\} \)

Stack element \( (\tau,m,i,\text{tried}) \)

\( \tau \): task 
\( m \): method 
\( i \): instruction pointer 
\( \text{tried} \): methods already tried
Progress (subroutine)

\[
\text{Progress}(\text{stack})
\]
\[
(\tau, m, i, \text{tried}) \leftarrow \text{top}(\text{stack})
\]
\[
\text{if } i \neq \text{nil} \text{ and } m[i] \text{ is a command then do}
\]
\[
\text{case status}(m[i])
\]
\[
\text{running: return}
\]
\[
\text{failure: Retry(stack); return}
\]
\[
\text{done: continue}
\]
\[
\text{if } i \text{ is the last step of } m \text{ then}
\]
\[
\text{pop}(\text{stack}) \quad \text{// remove stack’s top element}
\]
\[
\text{else do}
\]
\[
i \leftarrow \text{nextstep}(m, i)
\]
\[
\text{case type}(m[i])
\]
\[
\text{assignment: update } \xi \text{ according to } m[i]; \text{return}
\]
\[
\text{command: trigger command } m[i]; \text{return}
\]
\[
\text{task or goal: continue}
\]
\[
\tau' \leftarrow m[i]
\]
\[
\text{Candidates} \leftarrow \text{Instances}(\mathcal{M}, \tau', \xi)
\]
\[
\text{if } \text{Candidates} = \emptyset \text{ then Retry(stack)}
\]
\[
\text{else do}
\]
\[
\text{arbitrarily choose } m' \in \text{Candidates}
\]
\[
\text{stack} \leftarrow \text{push}((\tau', m', \text{nil}, \emptyset), \text{stack})
\]

Just a decision tree:

- $m[i]$ finished?
  - yes: return
  - no: retry
- more steps?
  - yes: pop stack
  - no:
    - if $C$ for $m[i]$?
      - yes: choose candidate $m'$
      - no: retry
    - otherwise: trigger it
Retry (subroutine)

Retry(\(stack\))

\((\tau, m, i, \text{tried}) \leftarrow \text{pop}(\text{stack})\)
\(\text{tried} \leftarrow \text{tried} \cup \{m\}\)
\(\text{Candidates} \leftarrow \text{Instances}(\mathcal{M}, \tau, \xi) \setminus \text{tried}\)
if \(\text{Candidates} \neq \emptyset\) then do
  arbitrarily choose \(m' \in \text{Candidates}\)
  \(\text{stack} \leftarrow \text{push}(\tau, m', \text{nil}, \text{tried}), \text{stack}\)\)
else do
  if \(\text{stack} \neq \emptyset\) then Retry(\(\text{stack}\))
  else do
    output(“failed to accomplish” \(\tau\))
    \(\text{Agenda} \leftarrow \text{Agenda} \setminus \text{stack}\)

Another decision tree:

pop top stack element \((\tau, m, i, \text{tried})\)
add \(m\) to \(\text{tried}\)

untried candidates for \(\tau\)?
  yes
  choose one
  push onto stack
  no

stack empty?
  yes
  fail
  no
  Retry

How does this differ from backtracking?
**Example**

- **Objects**
  - \( \text{Robots} = \{ \text{rbt} \} \)
  - \( \text{Containers} = \{ \text{c1}, \text{c2}, \ldots \} \)
  - \( \text{Locations} = \{ \text{loc1}, \text{loc2}, \ldots \} \)

- **State variables**
  - \( \text{loc}(r) \in \text{Locations} \)
  - \( \text{load}(r) \in \text{Containers} \cup \{ \text{nil} \} \)
  - \( \text{pos}(c) \in \text{Locations} \cup \text{Robots} \cup \{ \text{unknown} \} \)
  - \( \text{view}(l) \in \{ T, F \} \)
    - Whether the robot has looked at location \( l \) or not
    - If \( \text{view}(l) = T \) then for every container \( c \) at location \( l \), \( \text{pos}(c) = l \)

- **Commands to the execution platform:**
  - \( \text{take}(r,o,l) \): \( r \) takes object \( o \) at location \( l \)
  - \( \text{put}(r,o,l) \): \( r \) puts \( o \) at location \( l \)
  - \( \text{perceive}(r,l) \): robot \( r \) perceives what objects are at location \( l \)
Example

m-fetch($r,c$)
  task: $fetch(r,c)$
  pre: 
  body:
    if $pos(c) = unknown$ then
      $search(r,c)$
    else if $loc(r) = pos(c)$ then
      $take(r,c,pos(c))$
    else do
      $move-to(r,pos(c))$
      $take(r,c,pos(c))$

m-search($r,c$)
  task: $search(r,c)$
  pre: $pos(c) = unknown$
  body:
    if $\exists l (view(r,l) = F)$ then
      $move-to(r,l)$
      $perceive(l)$
      if $pos(c) = l$ then
        $take(r,c,l)$
      else $search(r,c)$
    else fail

Refinement stack

\[ \begin{align*}
\tau &: \quad fetch(r1,c2) \\
\textit{m} &: \quad ? \\
\textit{i} &: \quad (\text{see method}) \\
\textit{tried} &: \quad \emptyset
\end{align*} \]
**Example**

\[ \text{m-fetch}(r,c) \]
- **task:** fetch\((r,c)\)
- **pre:**
- **body:**
  - if pos\((c) = \text{unknown} \) then
    - search\((r,c)\)
  - else if loc\((r) = \text{pos}(c) \) then
    - take\((r,c,\text{pos}(c))\)
  - else do
    - move-to\((r,\text{pos}(c))\)
    - take\((r,c,\text{pos}(c))\)

\[ \text{m-search}(r,c) \]
- **task:** search\((r,c)\)
- **pre:** pos\((c) = \text{unknown} \)
- **body:**
  - if \(\exists l \ (\text{view}(r,l) = F) \) then
    - move-to\((r,l)\)
    - perceive\((l)\)
    - if pos\((c) = l \) then
      - take\((r,c,l)\)
    - else search\((r,c)\)
  - else fail

\( \tau: \) fetch\((r1,c2)\)
\( m: \) m-fetch\((r1,c2)\)
\( i: \) (see method)
\( \mathit{tried}: \) \( \emptyset \)
Example

m-fetch\((r,c)\)

- task: fetch\((r,c)\)
- pre:
- body:
  - if pos\((c)\) = unknown then
    - search\((r,c)\)
  - else if loc\((r)\) = pos\((c)\) then
    - take\((r,c,pos(c))\)
  - else do
    - move-to\((r,pos(c))\)
    - take\((r,c,pos(c))\)

m-search\((r,c)\)

- task: search\((r,c)\)
- pre: pos\((c)\) = unknown
- body:
  - if \(\exists l\) (view\((r,l)\) = F) then
    - move-to\((r,l)\)
    - perceive\((l)\)
      - if pos\((c)\) = l then
        - take\((r,c,l)\)
        - else search\((r,c)\)
    - else fail

\(\tau\): search\((r1,c2)\)
\(m\): ?
\(i\): (see method)
\(tried\): \(\emptyset\)

\(\tau\): fetch\((r1,c2)\)
\(m\): m-fetch\((r1,c2)\)
\(i\): (see method)
\(tried\): \(\emptyset\)
Example

m-fetch(\(r,c\))

- **task:** fetch(\(r,c\))
- **pre:**
- **body:**
  - if pos(c) = unknown then
    search(\(r,c\))
  - else if loc(r) = pos(c) then
    take(\(r,c,pos(c)\))
  - else do
    move-to(\(r,pos(c)\))
    take(\(r,c,pos(c)\))

m-search(\(r,c\))

- **task:** search(\(r,c\))
- **pre:** pos(c) = unknown
- **body:**
  - if \(\exists l \) (view(\(r,l\)) = F) then
    move-to(\(r,l\))
    perceive(l)
    if pos(c) = l then
      take(\(r,c,l\))
    else search(\(r,c\))
  - else fail

\(\tau\): search(\(r1,c2\))

\(m\): m-search(\(r1,c2\))

\(i\): (see method)

\(\text{tried}\): \(\emptyset\)

Refinement stack
m-fetch(r,c)
  task:  fetch(r,c)
  pre:
  body:
    if pos(c) = unknown then
      search(r,c)
    else if loc(r) = pos(c) then
      take(r,c,pos(c))
    else do
      move-to(r,pos(c))
      take(r,c,pos(c))

m-search(r,c)
  task:  search(r,c)
  pre:  pos(c) = unknown
  body:
    if \( \exists l \) (view(r,l) = F) then
      move-to(r,l)
      perceive(l)
      if pos(c) = l then
        take(r,c,l)
      else search(r,c)
    else fail

Example

\( \tau: \) search(r1,c2)
\( m: \) m-search(r1,c2)
\( i: \) (see method)
\( tried: \) \( \emptyset \)

\( \tau: \) fetch(r1,c2)
\( m: \) m-fetch(r1,c2)
\( i: \) (see method)
\( tried: \) \( \emptyset \)
Example

m-fetch(r,c)
  task: fetch(r,c)
  pre:
  body:
    if pos(c) = unknown then
      search(r,c)
    else if loc(r) = pos(c) then
      take(r,c,pos(c))
    else do
      move-to(r,pos(c))
      take(r,c,pos(c))

m-search(r,c)
  task: search(r,c)
  pre: pos(c) = unknown
  body:
    if ∃ l (view(r,l) = F) then
      move-to(r,l)
      perceive(l)
    if pos(c) = l then
      take(r,c,l)
    else search(r,c)
    else fail

Refinement stack

τ: search(r1,c2)
m: m-search(r1,c2)
i: (see method)
tried: ∅

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried: ∅
m-fetch(r,c)
  task: fetch(r,c)
  pre:
  body:
    if pos(c) = unknown then
      search(r,c)
    else if loc(r) = pos(c) then
      take(r,c,pos(c))
    else do
      move-to(r,pos(c))
      take(r,c,pos(c))

m-search(r,c)
  task: search(r,c)
  pre: pos(c) = unknown
  body:
    if \exists l (view(r,l) = F) then
      move-to(r,l)
      perceive(l)
    if pos(c) = l then
      take(r,c,l)
    else search(r,c)
    else fail

Example

\[ \tau: \text{search}(r1,c2) \]
\[ m: \text{m-search}(r1,c2) \]
\[ i: \text{(see method)} \]
\[ tried: \emptyset \]

\[ \tau: \text{fetch}(r1,c2) \]
\[ m: \text{m-fetch}(r1,c2) \]
\[ i: \text{(see method)} \]
\[ tried: \emptyset \]
Example

\(m\text{-fetch}(r,c)\)
- **Task:** fetch\((r,c)\)
- **Pre:**
- **Body:**
  - if pos\((c)\) = unknown then
    - search\((r,c)\)
  - else if loc\((r)\) = pos\((c)\) then
    - take\((r,c,\text{pos}(c))\)
  - else do
    - move-to\((r,\text{pos}(c))\)
    - take\((r,c,\text{pos}(c))\)

\(m\text{-search}(r,c)\)
- **Task:** search\((r,c)\)
- **Pre:** pos\((c)\) = unknown
- **Body:**
  - if \(\exists l\) (view\((r,l)\) = F) then
    - move-to\((r,l)\)
    - perceive\((l)\)
      - if pos\((c)\) = \(l\) then
        - take\((r,c,l)\)
      - else search\((r,c)\)
  - else fail

\(\tau\): search\((r1,c2)\)
\(m\): ?
\(i\): (see method)
\(\text{tried}\): \(\{m\text{-search}(r1,c2)\}\)

\(\tau\): fetch\((r1,c2)\)
\(m\): m-fetch\((r1,c2)\)
\(i\): (see method)
\(\text{tried}\): \(\emptyset\)

- If other candidates for search\((r1,c2)\), try them
- Not same as backtracking
  - Different current state

\(\text{Refinement stack}\)

\(\text{fetch(r1,c2)}\)
\(\text{search(r1,c2)}\)
\(\text{m-fetch(r1,c2)}\)
\(\text{m-search(r1,c2)}\)
\(\text{move-to(r1,loc1)}\)
\(\text{perceive(loc1)}\)

\(\text{sensor failure}\)

- If other candidates for search\((r1,c2)\), try them
- Not same as backtracking
  - Different current state
Example

\begin{align*}
\text{m-fetch}(r,c) \\
\text{task: } & \text{fetch}(r,c) \\
\text{pre: } & \\
\text{body: } & \\
\quad & \text{if } \text{pos}(c) = \text{unknown} \text{ then} \\
\quad & \quad \text{search}(r,c) \\
\quad & \text{else if } \text{loc}(r) = \text{pos}(c) \text{ then} \\
\quad & \quad \text{take}(r,c,\text{pos}(c)) \\
\quad & \text{else do} \\
\quad & \quad \text{move-to}(r,\text{pos}(c)) \\
\quad & \quad \text{take}(r,c,\text{pos}(c))
\end{align*}

\begin{align*}
\text{m-search}(r,c) \\
\text{task: } & \text{search}(r,c) \\
\text{pre: } & \text{pos}(c) = \text{unknown} \\
\text{body: } & \\
\quad & \text{if } \exists l \ (\text{view}(r,l) = \text{F}) \text{ then} \\
\quad & \quad \text{move-to}(r,l) \\
\quad & \quad \text{perceive}(l) \\
\quad & \quad \text{if } \text{pos}(c) = l \text{ then} \\
\quad & \quad \quad \text{take}(r,c,l) \\
\quad & \quad \text{else search}(r,c) \\
\quad & \text{else fail}
\end{align*}

\begin{align*}
\tau: & \text{fetch}(r1,c2) \\
\text{m: } & \text{m-fetch}(r1,c2) \\
\text{i: } & \text{(see method)} \\
\text{tried: } & \emptyset
\end{align*}
**Example**

\[ m\text{-fetch}(r,c) \]
- **task**: fetch \((r,c)\)
- **pre**: 
- **body**: 
  - if \(\text{pos}(c) = \text{unknown}\) then 
    - search \((r,c)\)
  - else if \(\text{loc}(r) = \text{pos}(c)\) then 
    - take \((r,c,\text{pos}(c))\)
  - else do 
    - move-to \((r,\text{pos}(c))\)
    - take \((r,c,\text{pos}(c))\)

\[ m\text{-search}(r,c) \]
- **task**: search \((r,c)\)
- **pre**: \(\text{pos}(c) = \text{unknown}\)
- **body**: 
  - if \(\exists l \ (\text{view}(r,l) = \text{F})\) then 
    - move-to \((r,l)\)
    - perceive \((l)\)
    - if \(\text{pos}(c) = l\) then 
      - take \((r,c,l)\)
    - else search \((r,c)\)
  - else fail

\[ \tau: \text{fetch}(r1,c2) \]
\[ m: ? \]
\[ i: \text{(see method)} \]
\[ \text{tried}: \{m\text{-fetch}(r1,c2)\} \]

- **Refinement stack**
- **Sensor failure**
  - If other candidates for fetch \((r1,c2)\), try them
  - Not same as backtracking
    - Different current state

Nau – Lecture slides for Automated Planning and Acting
Extensions to Rae

Events:

- method-name(\(arg_1, \ldots, arg_k\))
- event: event-identifier
- pre: test
- body: program

Example: an emergency

- If \(r\) isn’t already handing another emergency, then
  - stop what it’s doing
  - go handle the emergency

\[\text{Rae}(\mathcal{M})\]

\[\text{Agenda} \leftarrow \emptyset\]

loop

- until the input stream of external tasks and events is empty do
  - read \(\tau\) in the input stream
  - \(\text{Candidates} \leftarrow \text{Instances}(\mathcal{M}, \tau, \xi)\)
  - if \(\text{Candidates} = \emptyset\) then output("failed to acquire work")
  - else do
    - arbitrarily choose \(m \in \text{Candidates}\)
    - \(\text{Agenda} \leftarrow \text{Agenda} \cup \{\langle \tau, m, \text{nil, } \emptyset \rangle\}\)
    - for each \(\text{stack} \in \text{Agenda}\) do
      - \(\text{Progress} (\text{stack})\)
      - if \(\text{stack} = \emptyset\) then \(\text{Agenda} \leftarrow \text{Agenda} \setminus \{\text{stack}\}\)

- \(\text{m-emergency}(r,l,i)\)
  - event: emergency(\(l,i\))
  - pre: emergency-handling(\(r\)) = F
  - body: emergency-handling(\(r\)) \leftarrow T
    - if load(\(r\)) \neq \text{nil} then
      - put(\(r\),load(\(r\)))
      - move-to(\(l\))
      - address-emergency(\(l,i\))
Extensions to Rae

Goals:

- **method-name**\( (arg_1, \ldots, arg_k) \)
  - **task**: **achieve**(\( condition \))
  - **pre**: \textit{test}
  - **body**: **program**

- Write goal as a special kind of task
  - **achieve**(\( condition \))

- Like other tasks, but includes monitoring
  - **Modify Progress**
    - if \( condition \) becomes true before finishing body\( (m) \), stop early
    - if \( condition \) isn’t true after finishing body\( (m) \), fail and try another method
Extensions to Rae

- Concurrent subtasks:
  - refinement stack for each one

- Controlling the progress of tasks:
  - e.g., suspend a task for a while
  - If there are multiple stacks, which ones get higher priority?
    - Application-specific heuristics

- For a task $\tau$, which candidate to try first?
  - Refinement planning

---

body of a method:

\[
\ldots
\{\text{concurrent: } \tau_1, \tau_2, \ldots, \tau_n\}
\ldots
\]

Agenda = \{stack_1, stack_2, \ldots, stack_n\}

Candidates = \text{Instances}(\tau, M, \xi)
Summary

- **Representation:**
  - State variables
  - Commands to the execution platform
  - Refinement methods

- **Refinement Acting Engine (RAE)**
  - Purely reactive
    - For each task, event, or goal, select a method and apply it
  - Concurrent tasks
    - For each, a refinement stack
  - If something goes wrong, retry with a different method
    - Different from backtracking