Hybrid Architecture

In this project we plan to use Hybrid Architecture; it is a combination of both reactive and deliberative planning at different levels.

Refinement Acting Engine (RAE)

Reactive part of the plan is implemented using RAE (Refinement Acting Engine) which is purely reactive and has no Run-lookahead planning procedure.

Input to RAE: External tasks/events, and current state of the system.

Output from RAE: Command to the execution platform to accomplish the task/event.

For RAE, task can be any operation necessary for the system to achieve a desired state in our case it can be picking an object, placing an object etc. For each task in the Agenda we form a refinement stack which has different methods, control structures, assignment statements, commands to execution platform and Booleans necessary to accomplish the task/event.

The basic algorithm for RAE would look as described below:

```
Agenda = {current refinement stacks}
loop:
    if new external tasks/events then
        for each one, add a refinement stack to Agenda
    for each stack in Agenda
        Progress it, and remove it if it's finished
```

Fig 1 Pseudocode for RAE [1]

Here "Agenda" contains the current refinement stack for all the tasks needed to be achieved for accomplishing the stated goal, if a new external task/event is encountered it is added to Agenda and then "Progress" subroutine is called and its execution stops when the task is finished/failure is encountered. If the progress execution results in failure, then "Retry" subroutine is called which takes a new method (if available) that can achieve the given task. However, if no such alternative method exists then it returns failure. The way we will implement progress and retry is mentioned below in pseudo code.

```
Progress(stack)
  (τ, m, i, tried) ← top(stack)
  if i ≠ nil and m[i] is a command then do
     case status(m[i])
        running: return
        failure: Retry(stack); return
        done: continue
  if i is the last step of m then
        pop(stack)    // remove stack's top element
  else do
     i ← nextstep(m, i)
     case type(m[i])
        assignment: update ξ according to m[i]; return
        command: trigger command m[i]; return
        task or goal: continue
     τ' ← m[i]
     Candidates ← Instances(M, τ', ξ)
     if Candidates = ∅ then Retry(stack)
     else do
        arbitrarily choose m' ∈ Candidates
        stack ← push((τ', m', nil, ∅),stack)
```

Fig 2 Progress subroutine [1]

```
Retry(stack)
  (τ, m, i, tried) ← pop(stack)
  tried ← tried ∪ {m}
  Candidates ← Instances(M, τ, ξ)\ tried
  if Candidates≠ ∅ then do
     arbitrarily choose m' ∈ Candidates
        stack ← push((τ, m', nil, tried),stack
  else do
     if stack≠ ∅ then Retry(stack)
     else do
        output("failed to accomplish" τ)
        Agenda ← Agenda\stack
```

Fig 3 Retry subroutine [1]

Here we observe that in progress and retry we select method candidate arbitrarily and this can result in bad outcomes. Thus, a planned selection of method can help overcoming unrecoverable failures, costly solution, and can reduce computation time.

Thus, to introduce a deliberative planning component to RAE and to make the selection of a candidate method planned. We can use a "RAE-Plan". Where, when we need to select a method, we try all different actions/method by running them in parallel on multi-threads in a simulation environment and pass the action which results in success to the execution platform. The new pseudo code is mentioned below:

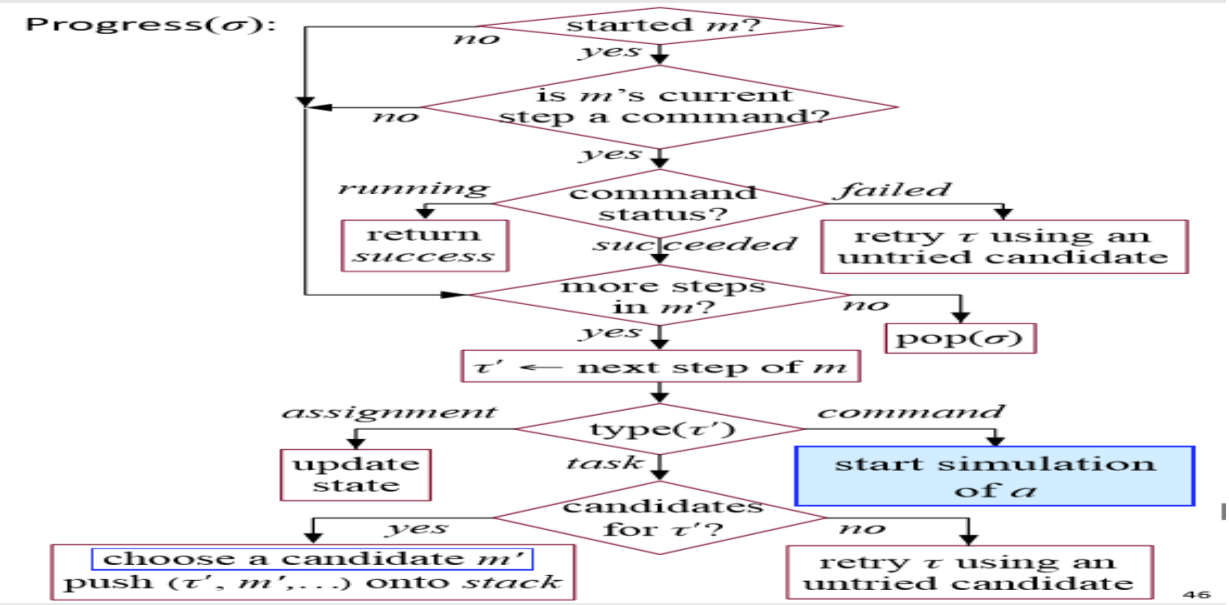

Fig 4 RAE-Plan [1]

Our Block Diagram:

The block diagram represents the task of “Building Kit”.

The main task “Build Kit” can be divided into sub tasks:

1) Pick-up Order
2) Environment Model
3) Pick-up Part
4) Placing Part

Here the whole implementation can be categorized as follows:

Perception and modelling: 1) Environment Model

Planning: 1) Pick-up Order

2) Pick-up Part

3) Placing Part

Execution: 1) Gripper Action

2) Robot Movement

Pick-up Order:

1) It has two sub tasks Pick up from bin and pick-up from conveyor.
2) Each has two components the type of part and part location.
3) The two components mentioned in (2) above requires sensor system which provides the needed information on which further processing like frame transformation etc. is performed.

Environment Modeling:

1) It has sensor system, Robot and AGVs
2) Sensor system has all the sensors used for sensing the environment to accomplishing the task.
3) Robot has 2 robots namely robot 1 and robot 2 which are used in the task, each robot has information about its location and pose, and gripper.
4) Gripper has information about the end-effector and its current state(on/off).
5) AGVs have information about its tray location (AGVs location) and AGV status which tells the number of parts on tray and other necessary data.

Pick-up Part:

1) It gets the information about the part needed to be picked from Pick-up Order.
2) It gets the information about the current robot status from the Environment Model.
3) It takes the robot to the required location and pose without collision with other robots and static obstacles which include sensors, bins etc.
4) Sub task gripper action helps in turning on/off the gripper when needed.

Placing Part:

1) It gets the information about the part needed to be placed from Pick-up Order.
2) It gets the information about the current robot status from the Environment Model.
3) It takes the robot to the required location and pose without collision with other robots and static obstacles which include sensors, bins etc.
4) It has a sub task to check for faulty part. If the part is faulty it picks the part from the tray and throws the faulty part mid-way and pick the next good part using Pick-up Part.
5) Sub task gripper action helps in turning on/off the gripper when needed.

We can send the tasks to RAE-plan and implement the Hybrid Architecture.

References:

1) CMSC-722 lecture notes and Automated Planning and Acting.