Robust Task Execution: RAPS [Firby PhD]

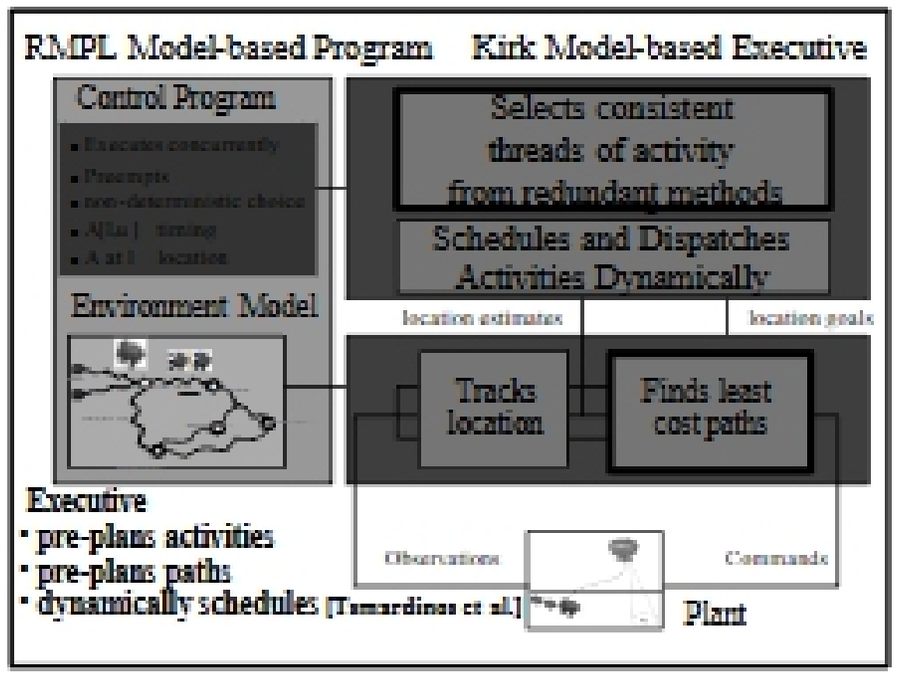
- RAPS Exploits contingencies by performing functionally redundant method selection
  - Methods are chosen based on the current situation.
  - If a method fails, another is tried instead.
  - Tasks do not complete until satisfied.
  - Methods can include monitoring subtasks to deal with contingencies and opportunities.

➢ **Methods selected reactively**

➟ **Model-predictive dispatch**

➢ **Goals explicitly observable and controllable**

➟ **Model-based execution**

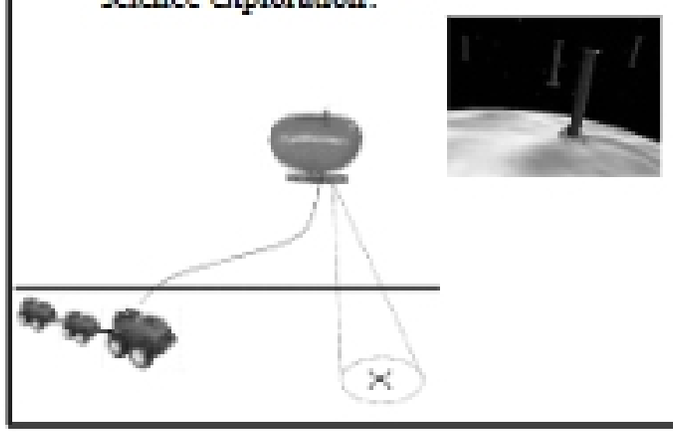


Outline

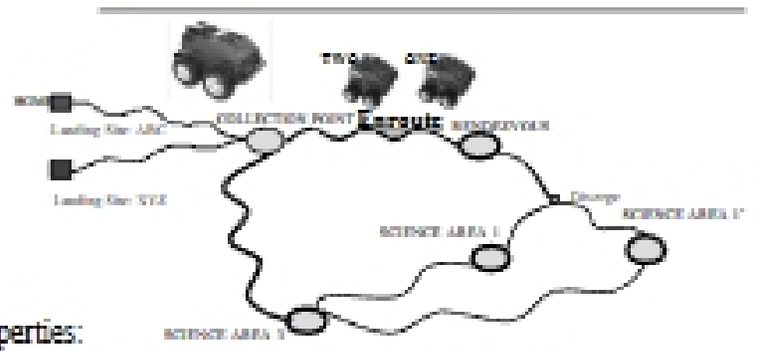
- Safe Procedural Execution
- Model-Predictive Dispatch
  - Model-based Programming
  - Temporal Plan Networks (TPN)
  - Activity Planning (Kirk)
  - Unifying Activity and Path Planning
- Model-based Reactive Planning

Example: Cooperative Mars Exploration

How do we coordinate heterogeneous teams of orbiters, rovers and air vehicles to perform globally optimal science exploration?



Example: Cooperative Mars Exploration



Properties:

- Teams exploit a hierarchy of complex strategies.
- Maneuvers are temporally coordinated.
- Novel events occur during critical phases.
- Quick responses draw upon a library of contingencies.
- Selected contingencies must respect timing constraints.

Reactive Model-based Programming

Idea: Describe team behaviors by starting with a rich concurrent, embedded programming language (RMPL, TCC, Esterel):

- c
- If c next A
- Unless c next A
- A, B
- Always A
- Sensing/actuation activities
- Conditional execution
- Preemption
- Full concurrency
- Iteration

Add temporal constraints:

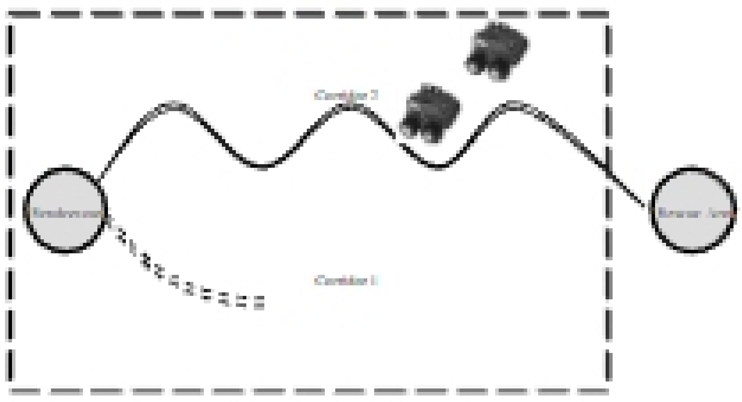
- A [l,u]
- Timing

Add choice (non-deterministic or decision-theoretic):

- Choose {A, B}
- Contingency

Example Enroute Activity:

Enroute



RMPL for Group-Enroute

```
Group-Enroute()[l,u] = {
  choose {
    do {
      Group-Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH1_OK,
    do {
      Group-Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH2_OK
  };
  {
    Group-Transmit(OPS,ARRIVED)[0,2],
    do {
      Group-Wait(HOLD1,HOLD2)[0,u*10%]
    } watching PROCEED
  }
}
```

RMPL for Group-Enroute

Activities:

```
Group-Enroute()[l,u] = {
  choose {
    do {
      Group-Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH1_OK,
    do {
      Group-Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH2_OK
  };
  {
    Group-Transmit(OPS,ARRIVED)[0,2],
    do {
      Group-Wait(HOLD1,HOLD2)[0,u*10%]
    } watching PROCEED
  }
}
```

RMPL for Group-Enroute

Conditionality and Preemption:

```
Group-Enroute()[l,u] = {
  choose {
    do {
      Group-Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH1_OK,
    do {
      Group-Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH2_OK
  };
  {
    Group-Transmit(OPS,ARRIVED)[0,2],
    do {
      Group-Wait(HOLD1,HOLD2)[0,u*10%]
    } watching PROCEED
  }
}
```

RMPL for Group-Enroute

Sequentiality:
Concurrency:

```
Group-Enroute()[l,u] = {
  choose {
    do {
      Group-Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH1_OK,
    do {
      Group-Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH2_OK
  };
  {
    Group-Transmit(OPS,ARRIVED)[0,2],
    do {
      Group-Wait(HOLD1,HOLD2)[0,u*10%]
    } watching PROCEED
  }
}
```

RMPL for Group-Enroute

Temporal Constraints:

```
Group-Enroute()[l,u] = {
  choose {
    do {
      Group-Fly-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH1_OK,
    do {
      Group-Fly-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
    } maintaining PATH2_OK
  };
  {
    Group-Transmit(OPS,ARRIVED)[0,2],
    do {
      Group-Wait(HOLD1,HOLD2)[0,u*10%]
    } watching PROCEED
  }
}
```