Model-based Programming of Robotic Space Explorers

mers.csail.mit.edu

Brian C. Williams
16.410 / 16.413
October 18th, 2010

Outline

- Robotic Exploration
- Model-based Programming and Execution

Robotic Spacecraft Require Large Human Teams to Operate

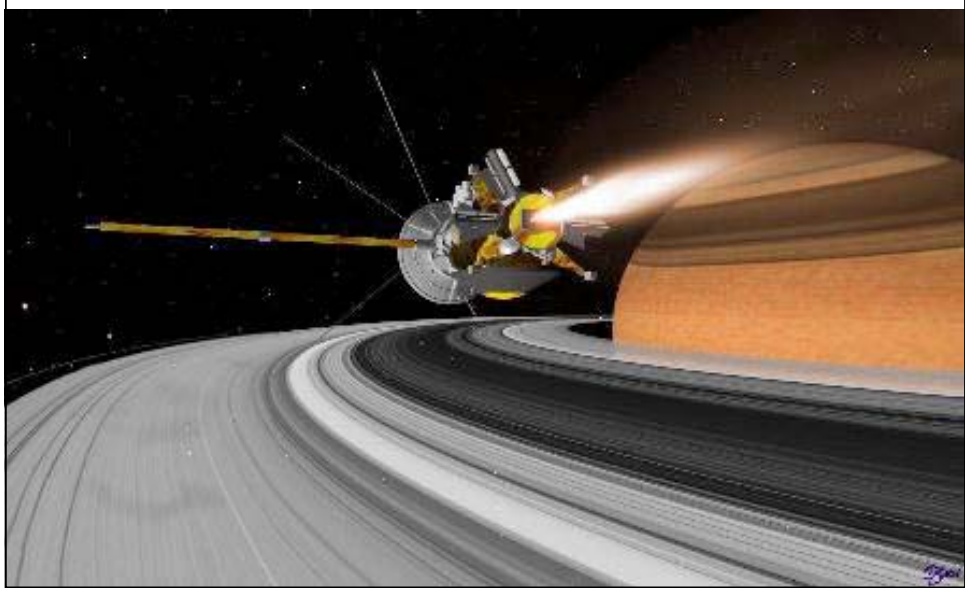
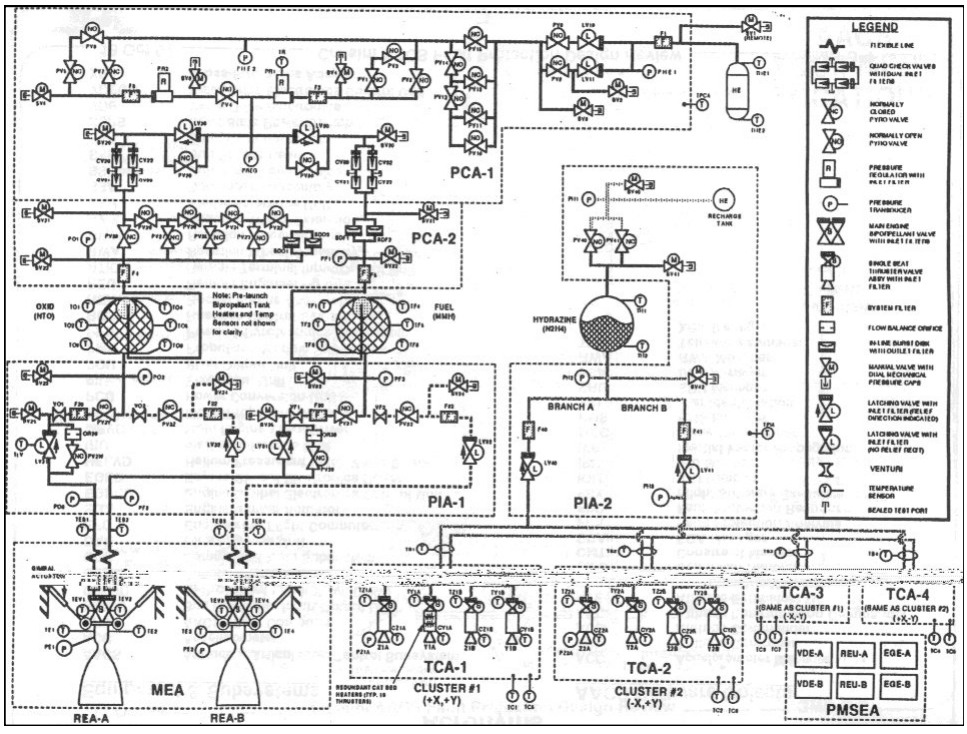


Image credit: NASA.



But They Still Fail

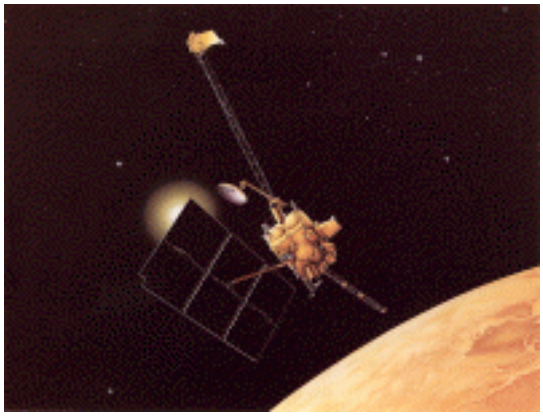
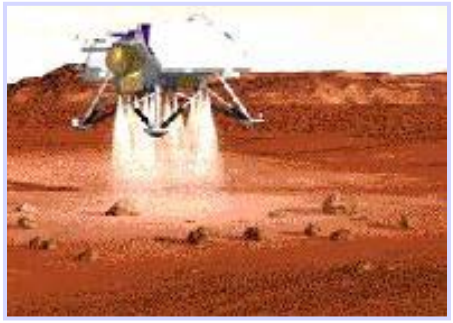


Image credit: NASA.

Mars Observer

Mars Polar Lander Failure



Mars Polar Lander Failure

Leading Diagnosis:

- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 50ft.
- Begins polling leg monitors to determine touch down.
- Latched noise spike read as touchdown.
- Engine shutdown at ~50ft.

Programmers are overwhelmed by the bookkeeping of reasoning about unlikely hidden states

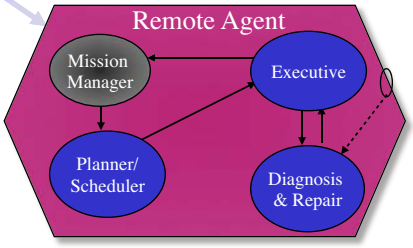
Fault Aware Systems:
Create executives
That reason and coordinate
on the fly from models

Outline

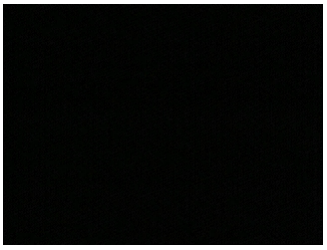
- Robotic Exploration
- Model-based Programming and Execution

1999:Remote Agent on Deep Space One

~~Goals~~

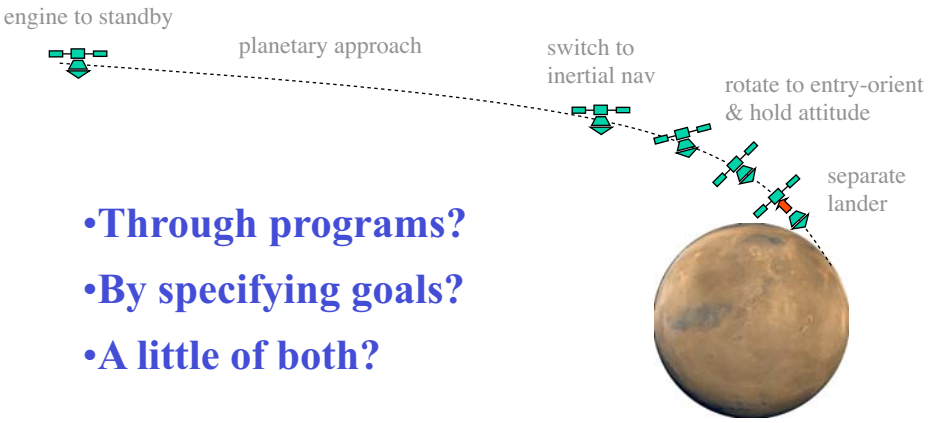


[Williams & Nayak, AAAI 95; Muscettola et al, AIJ 00]



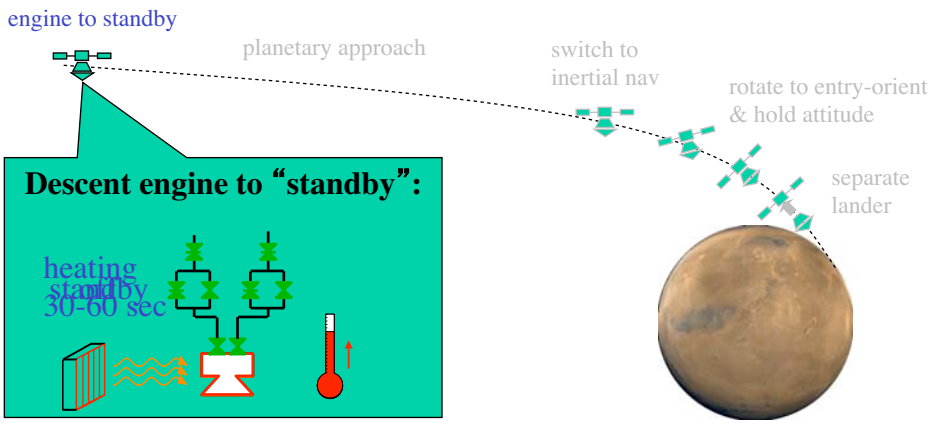
1. Commanded by giving goals
2. Closes loop on goals
3. Reasons from commonsense models

How Should Engineers Guide Model-based Executives?

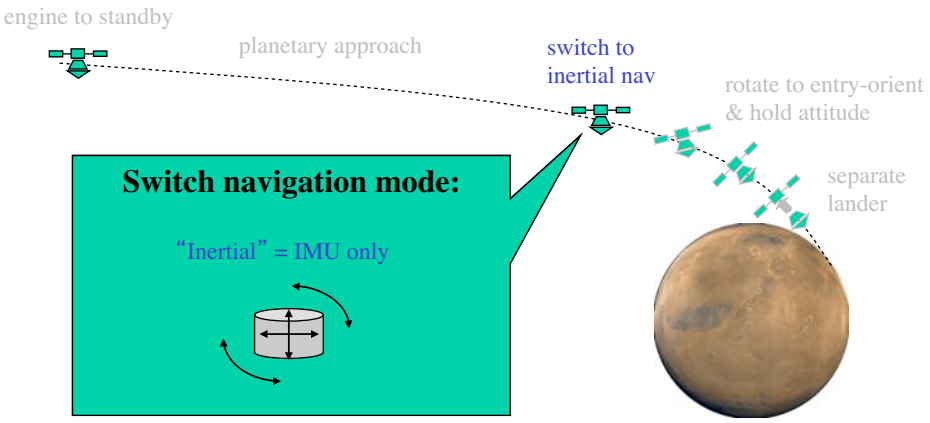


- **Through programs?**
- **By specifying goals?**
- **A little of both?**

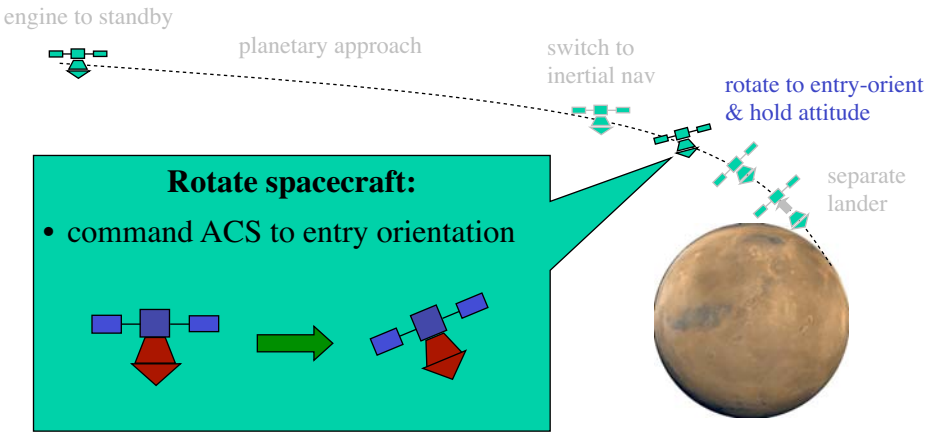
Mission Storyboards Specify Evolving States



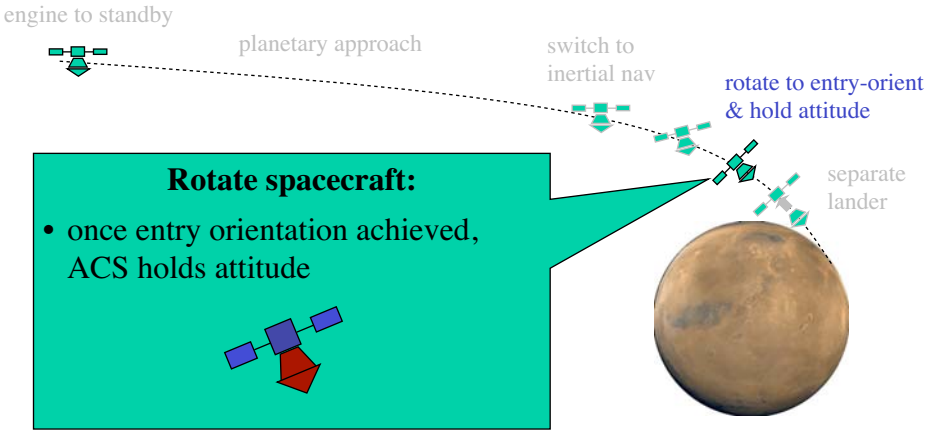
Mission Storyboards
Specify Evolving States



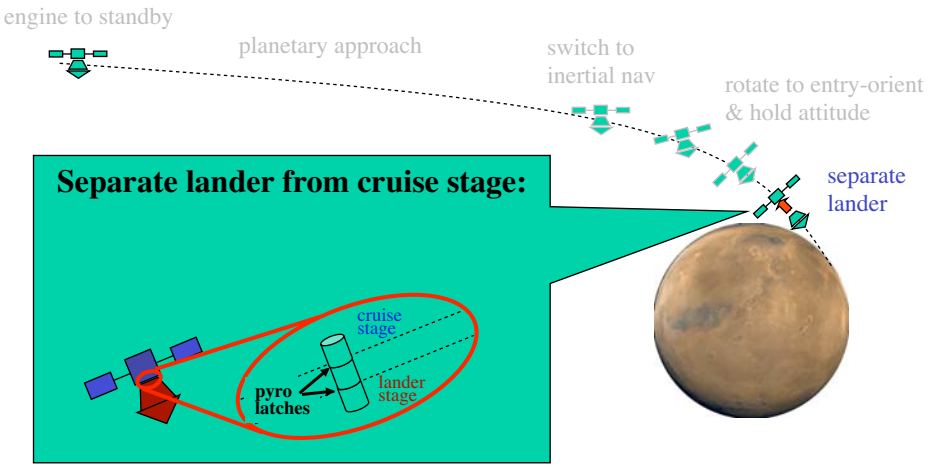
Mission Storyboards
Specify Evolving States



Mission Storyboards
Specify Evolving States



Mission Storyboards
Specify Evolving States



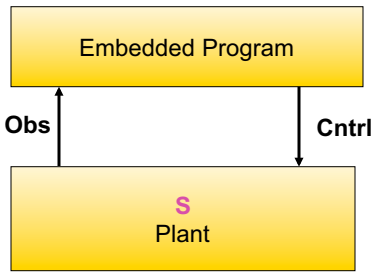
Model-based Programs, Like Storyboards, Specify the Evolution of Abstract States

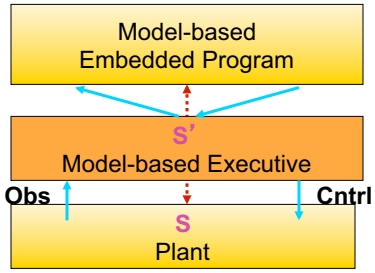
Embedded programs:

- Read sensors
- Set actuators

Model-based programs:

- Read abstract state
- Write abstract state





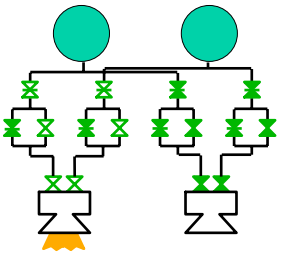
Model-based Program =
Control Program on "State"
+ Plant Model

Model-based executives map between state, sensors & actuators.

[Williams et al, IEEE Proc 02]

Model-based Programming of a Saturn Orbiter

Turn camera off and engine on

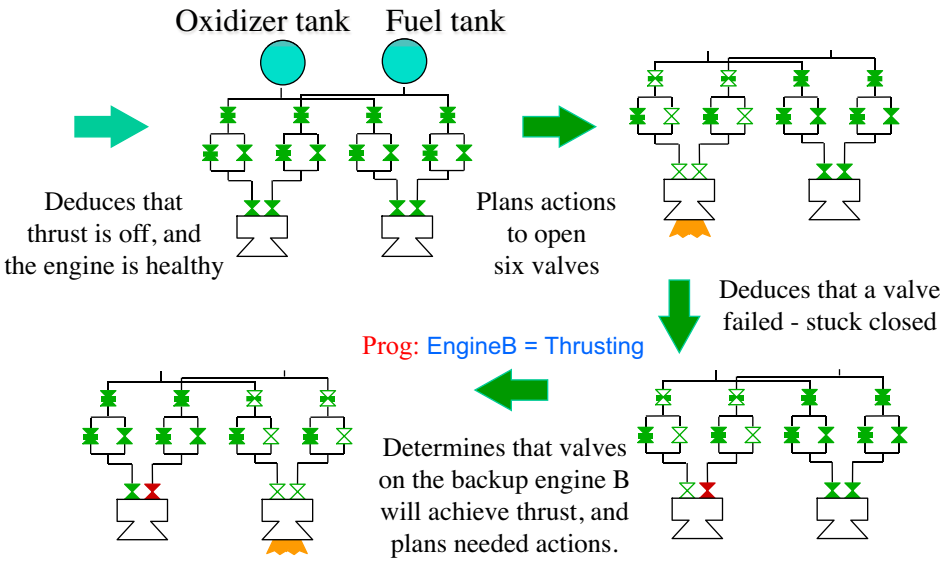


Science Camera

```
OrbitInsert()::

do-watching (EngineA = Thrusting OR
             EngineB = Thrusting)
   parallel {
      EngineA = Standby;
      EngineB = Standby;
      Camera = Off;
      do-watching (EngineA = Failed)
         {when-donext (EngineA = Standby) AND
                      Camera = Off)
            EngineA = Thrusting};
      when-donext (EngineA = Failed AND
                   EngineB = Standby AND
                   Camera = Off)
         EngineB = Thrusting}
```

The program assigns EngineA = Thrusting,
and the model-based executive . . . .



Embedded Program

Expressions:

1. s — Conditions on sensors
2. u — Assignments to control variables

Control constructs:

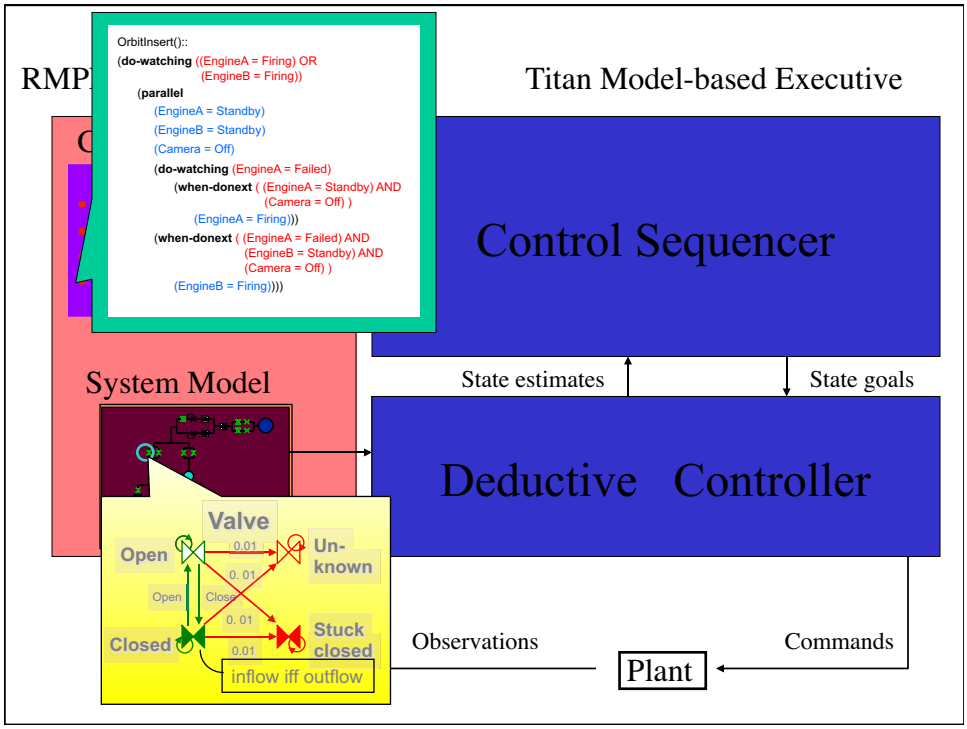
1. u — Control assignments
2. If s next A — Conditional execution
3. Unless s next A — Preemption
4. A, B — Full concurrency
5. Always A — Iteration

Reactive Model-based Program

Idea: A concurrent constraint program (e.g. TCC/HCC [Saraswat et al.])

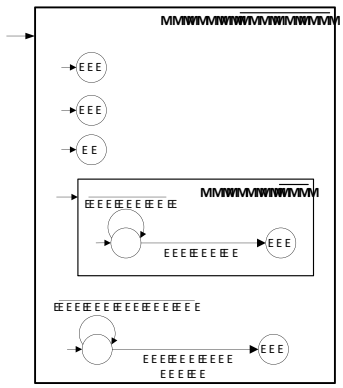
- whose constraints c operate on the state of the plant s, and
- replaces the constraint store with a model-based controller:

1. c[s]	Primitive constraint on state
2. If c[s] next A	Conditional execution
3. Unless c[s] next A	Preemption
4. A, B	Full concurrency
5. Always A	Iteration



RMPL Semantics is in Terms of Constraint Automata

```
OrbitInsert()::
  do-watching (EngineA = Thrusting OR
               EngineB = Thrusting)
    in-parallel {
      EngineA = Standby,
      EngineB = Standby,
      Camera = Off,
      do-watching (EngineA = Failed)
        when-donext (EngineA = Standby AND
                     Camera = Off )
          EngineA = Thrusting,
      when-donext ( EngineA = Failed AND
                    EngineB = Standby AND
                    Camera = Off )
        EngineB = Thrusting
    }
```



- Automata are hierarchical.
- Automata locations and transition guards have associated constraints on plant state s.

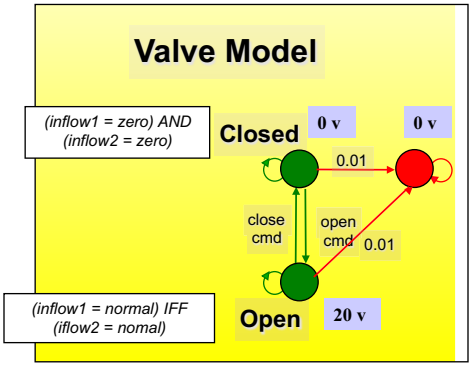
Components are Modeled using Probabilistic Constraint Automata

component modes…

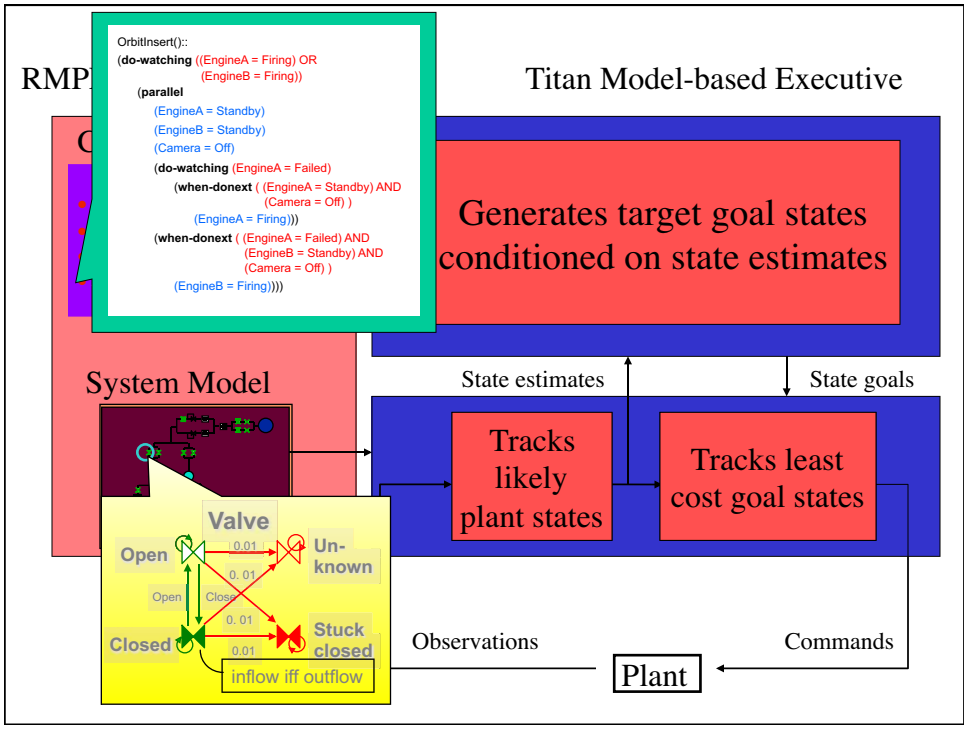
described by logical constraints on variables…

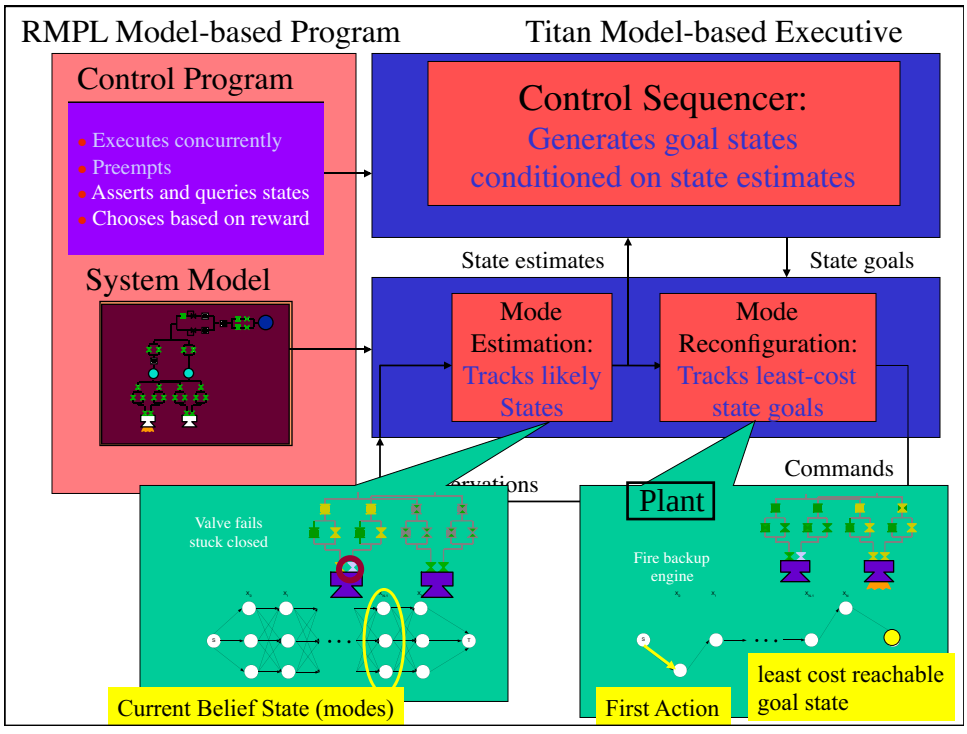
deterministic and probabilistic transitions

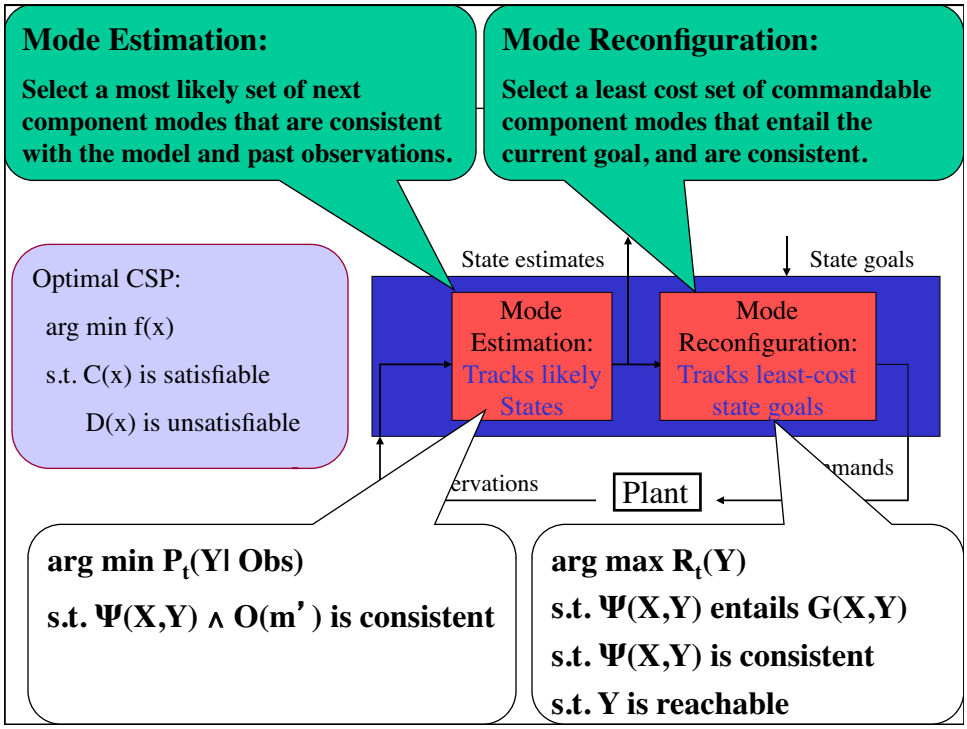
cost/reward

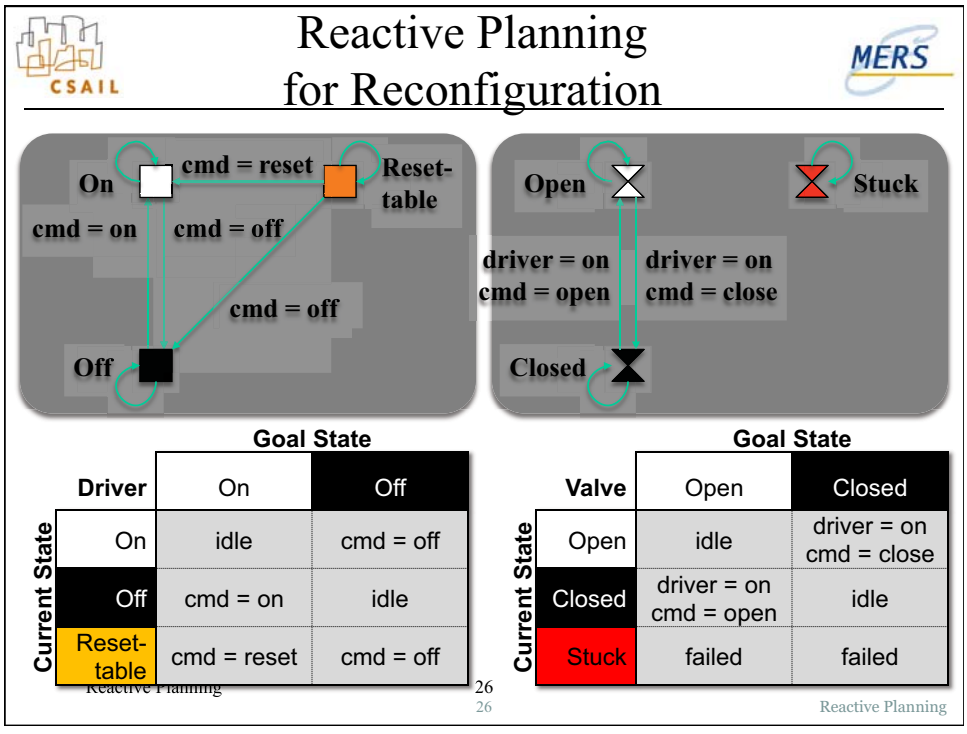


Logic/Constraints + Markov Processes + Concurrency

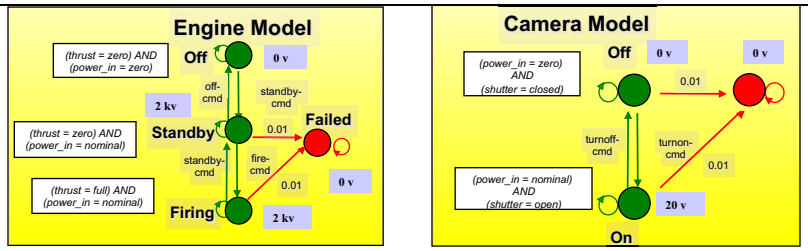

RMP
Titan Model-based Executive
OrbitInsert()::
(do-watching ((EngineA = Firing) OR
(EngineB = Firing))
(parallel
(EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed)
(when-donext ((EngineA = Standby) AND
(Camera = Off))
(EngineA = Firing)))
(when-donext ((EngineA = Failed) AND
(EngineB = Standby) AND
(Camera = Off))
(EngineB = Firing))))
Generates target goal states conditioned on state estimates
System Model
State estimates
State goals
Tracks likely plant states
Tracks least cost goal states
Valve
Open
Un-known
Closed
Stuck closed
inflow iff outflow
Observations
Commands
Plant


RMPL Model-based Program
Titan Model-based Executive
Control Program
Executes concurrently
Preempts
Asserts and queries states
Chooses based on reward
System Model
Control Sequencer:
Generates goal states conditioned on state estimates
State estimates
State goals
Mode Estimation:
Tracks likely States
Mode Reconfiguration:
Tracks least-cost state goals
Commands
Valve fails stuck closed
Plant
Fire backup engine
Current Belief State (modes)
First Action
least cost reachable goal state


Mode Estimation:
Select a most likely set of next component modes that are consistent with the model and past observations.
Mode Reconfiguration:
Select a least cost set of commandable component modes that entail the current goal, and are consistent.
Optimal CSP:
arg min f(x)
s.t. C(x) is satisfiable
D(x) is unsatisfiable
State estimates
State goals
Mode Estimation: Tracks likely States
Mode Reconfiguration: Tracks least-cost state goals
ervations
mands
Plant
arg min P_t(Y| Obs)
s.t. Ψ(X,Y) ∧ O(m') is consistent
arg max R_t(Y)
s.t. Ψ(X,Y) entails G(X,Y)
s.t. Ψ(X,Y) is consistent
s.t. Y is reachable


CSAIL
Reactive Planning for Reconfiguration
MERS
On
cmd = reset
Reset-table
cmd = on
cmd = off
cmd = off
Off
Open
Stuck
driver = on
cmd = open
driver = on
cmd = close
Closed
Goal State
Driver
On
Off
Current State
On
idle
cmd = off
Off
cmd = on
idle
Reset-table
cmd = reset
cmd = off
Goal State
Valve
Open
Closed
Current State
Open
idle
driver = on
cmd = close
Closed
driver = on
cmd = open
idle
Stuck
failed
failed
Reactive Planning
26
26
Reactive Planning

Variants on Probabilistic Constraint Automata define a Family of RMPL Languages



- Complex, discrete behaviors
  - modeled through concurrency, hierarchy and timed transitions.
- Anomalies and uncertainty
  - modeled by probabilistic transitions
- Physical interactions
  - modeled by discrete and continuous constraints

Model-based Programming of Embedded Systems

- To survive decades embedded systems orchestrate complex regulatory and immune systems.
- Future systems will be programmed with models, describing themselves and their environments.
- Runtime kernels will be agile, deducing and planning by solving optimization problems with propositional constraints.
- Model-based reactive planners respond quickly to failure, while using compile-time analysis of structure to respond quickly and concisely to indirect effects.

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010