Model Based Programming

September 17th, 2010

James Marshall

B.C. Williams, M. D. Ingham, S. H. Chung, P. H. Elliot.
Model-Based Programming of Intelligent Embedded Systems and Robotic Space Explorers

P. Kim, B. C. Williams, M. Abramson
Executing Reactive, Model-based Programs through Graph-based Temporal Planning

Overview

- Reactive Model-Based Programming Language (RMPL)
- Titan Model-base Executive
  - Extended with Kirk, the temporal planner

Motivation

- Low level system interactions are simply too complicated and numerous to be handled explicitly by programmers
- These were the causes of failure with the Mars Polar Lander and Mars Climate Orbiter
- Problem will be exacerbated as embedded systems continue to become more complicated

Reactive Synchronous Lanuage

- Esterel, Lustre, and SIGNAL
- Explicit concurrent and sequential statements
- Precise time model
- Handles sensors and actuators well

Model Based Programming

- Built on reactive (real-time) synchronous languages
- Programmer defines a physical model (plant model) and a control program
- Control program is written at a state level
- Obscures the measured state of the physical system from the programmer, and deduces how to reach the desired state

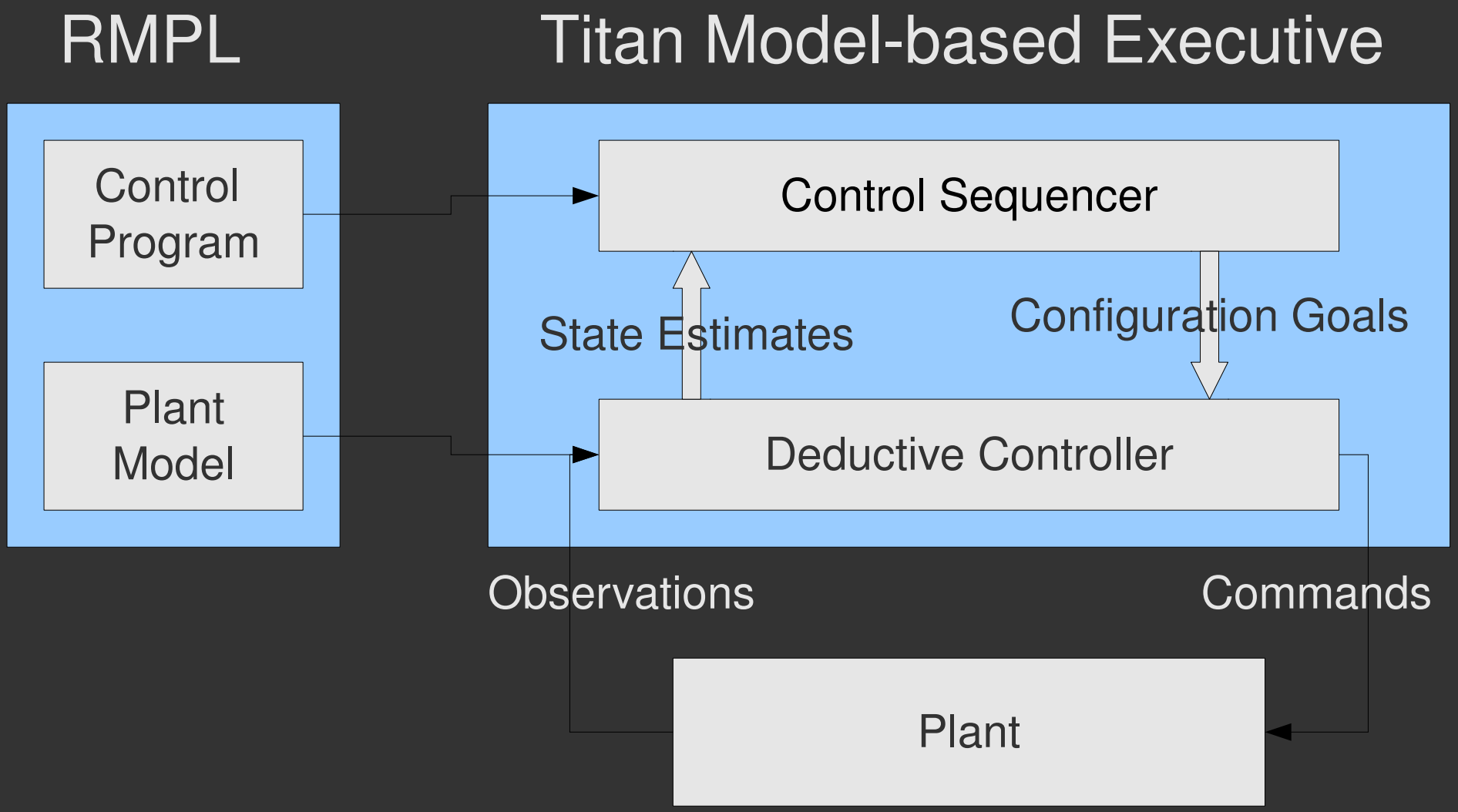
Reasoning at a State Level

- Embedded programmers do this anyway
- States and transitions described by Plant Model
- Should not be bogged down with details
  - Measuring state (dealing with noise, sensor failure)
  - Selecting set of transitions (depends on failures, time constraints, etc.)
  - Checking that desired state has been reached

Engine Example

```
Launch() :: {
   do {
      EngineA = Standby,
      EngineB = Standby,
      do {
         when EngineA = Standby
            donext EngineA = Firing
      } watching EngineA = Failed,
      when EngineA = Failed && EngineB = Standby
         donext EngineB = Firing
   } watching EngineA = Firing || EngineB = Firing
}
```

Architecture Model



RMPL

- Expression →

  assertion | combinator | program_invocation
- program_invocation → program_name(arglist)
- Combinator →

  A **maintaining** *c* | **do** *A* **watching** *c* |

  if *c* **thennext** *A* | **unless** *c* **thennext** *A* |

  A, *B* | *A*; *B* | **always** *A*
- *A* is an expression, *c* is a constraint

RMPL Continued

- Assertion: an achieve constraint
- Notice that we are describing:
  - desired states for certain conditions
  - Sequences of state transitions (state evolutions)
- The transitions themselves are not described

Plant Model

- Defines states and transitions
- Built up from component models
  - Modes, constraints, and probabilistic transitions
- Probabilistic transitions
  - Allows for unanticipated failure modes
  - Makes everything much more complicated

Markov whats?

- Plant model translated into a:
- Partially observable Markov decision process (POMDP)
  - Form a mapping between ideal state and actual state
  - Discover the best set of transitions between two states
- Allows the true state to be represented as a probability distribution, based on the observable properties

Markov Power

- POMDP deals with (hides) all of these problems:
  - Signal noise
  - Probabilistic transitions (failures!)
  - Incomplete, insufficient, or faulty sensor data
  - Uncertainty with actuators

HCA

- Encode RPML into Hierarchical Constraint Automata (HCA)
- Each HCA:
  - Composes sets of concurrently running automata
  - Has a goal constraint
  - Has a maintenance constraint
  - Hierarchical
- Similar to Esterel

Control Sequencer

- Measures estimated state from Deductive Controller
- Composes desired state from live HCAs
  - Checks maintenance constraints
  - Uses goal constraints
- Issues single command to Deductive Controller to move towards desired state
- Repeat.

That's all, for now

- How does the temporal planner fit?
- More complicated example, whole system
- Questions?