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.

© Source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/fairuse.
But They Still Fail

Mars Observer

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.

Mars Polar Lander Failure

Image credit: NASA.

© Source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/fairuse.
Robotic Exploration
Model-based Programming and Execution

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

[Williams & Nayak, AAAI 95; Muscella et al, AIJ 00]
How Should Engineers Guide Model-based Executives?

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

Mission Storyboards
Specify Evolving States

Descent engine to "standby":
Engine to standby
Planetary approach
Switch to inertial nav
Rotate to entry-orient & hold attitude
Separate lander

Heating
30-60 sec
Switch navigation mode:

“Inertial” = IMU only

Rotate spacecraft:

• command ACS to entry orientation
Mission Storyboards
Specify Evolving States

engine to standby

planetary approach

switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

**Rotate spacecraft:**
- once entry orientation achieved, ACS holds attitude

**Separate lander from cruise stage:**
- pyro latches
- cruise stage
- lander stage
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

OrbitInsert():

\[
\begin{align*}
\text{do-watching} & \quad (\text{EngineA = Thrusting OR EngineB = Thrusting}) \\
\text{parallel} & \quad (\text{EngineA = Standby; EngineB = Standby; Camera = Off;}) \\
\text{do-watching} & \quad (\text{EngineA = Failed}) \\
\text{when-donext} & \quad (\text{EngineA = Standby AND Camera = Off}) \\
\text{EngineA = Thrusting};
\end{align*}
\]

\[
\begin{align*}
\text{when-donext} & \quad (\text{EngineA = Failed AND EngineB = Standby AND Camera = Off}) \\
\text{EngineB = Thrusting}.
\end{align*}
\]
The program assigns \texttt{EngineA = Thrusting},
and the model-based executive . . . .

Deduces that thrust is off, and
the engine is healthy

Deduces that a valve
failed - stuck closed

Plans actions
to open six valves

Determines that valves
on the backup engine B
will achieve thrust, and
plans needed actions.

\texttt{Prog: EngineB = Thrusting}

\begin{itemize}
  \item \texttt{u} Conditions on sensors
  \item \texttt{s} Assignments to control variables
\end{itemize}

\begin{itemize}
  \item Control assignments
  \item Conditional execution
  \item Preemption
  \item Full concurrency
  \item Iteration
\end{itemize}
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

Titan Model-based Executive

Control Sequencer

Deductive Controller

RMP

System Model

Valve

Open

Close

Unknown

Stuck closed

inflow iff outflow

Observations

State estimates

State goals

Commands

Plant

OrbitInsert():
(do-watching (EngineA = Firing) OR (EngineB = Firing))
(parallel)
(EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed))
(when-donext (EngineA = Standby)
(Camera = Off))
(EngineA = Firing))
(when-donext (EngineA = Failed)
(EngineB = Standby)
(Camera = Off))
(EngineB = Firing))

RMPL Semantics is in Terms of Constraint Automata

- 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
Titan Model-based Executive

Generates target goal states conditioned on state estimates

State estimates

State goals

Tracks likely plant states

Tracks least-cost goal states

Observations

Commands

Plant

System Model

Valve

Opened

Closed

Unknown

Open

Closed

Stuck closed

Valve fails

stuck closed

Current Belief State (modes)

RMPL Model-based Program

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

Observations

Commands

Plant

least cost reachable goal state

First Action

Fire backup engine

S

T

X0 X1 XN-1 XN

least cost reachable goal state

First Action

Fire backup engine

S

T

X0 X1 XN-1 XN
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:
\[ \text{arg min } f(x) \]
\[ \text{s.t. } C(x) \text{ is satisfiable} \]
\[ \text{D(x) is unsatisfiable} \]

\[ \text{arg min } P_t(Y|\text{Obs}) \]
\[ \text{s.t. } \Psi(X,Y) \land O(m') \text{ is consistent} \]

\[ \text{arg max } R_c(Y) \]
\[ \text{s.t. } \Psi(X,Y) \text{ entails } G(X,Y) \]
\[ \text{s.t. } \Psi(X,Y) \text{ is consistent} \]
\[ \text{s.t. } Y \text{ is reachable} \]

Reactive Planning for Reconfiguration

<table>
<thead>
<tr>
<th>Goal State</th>
<th>Driver</th>
<th>On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cmd = on</td>
<td>cmd = off</td>
<td>cmd = off</td>
</tr>
<tr>
<td>On</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Off</td>
<td>reset</td>
<td>cmd = off</td>
<td>cmd = off</td>
</tr>
<tr>
<td>Reset-table</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal State</th>
<th>Valve</th>
<th>Open</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>driver = on</td>
<td>driver = on</td>
<td>cmd = close</td>
</tr>
<tr>
<td>Open</td>
<td>close</td>
<td>cmd = open</td>
<td>cmd = close</td>
</tr>
<tr>
<td>Closed</td>
<td></td>
<td>cmd = open</td>
<td></td>
</tr>
<tr>
<td>Stuck</td>
<td>failed</td>
<td>cmd = open</td>
<td>failed</td>
</tr>
</tbody>
</table>

Driver
Current State
On
Off
Reset-table

Valve
Current State
Open
Closed

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

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.