A Reactive Model-based Programming Language for Robotic Space Explorers

Michel Ingham, Robert Ragno, Brian Williams
MIT Space Systems Lab
MIT Artificial Intelligence Lab
Cambridge, MA

i-SAIRAS
June 21, 2001

Outline

• Motivation, Objective & Approach
• Example Scenario
• Introduction to Model-based Programming
• Reactive Model-based Programming Language (RMPL) Overview
• Compilation and Execution of Model-based Programs
• Future work

Motivation for Highly Autonomous Systems

State-of-the-art Autonomy S/W

- Barrier to wide deployment of autonomy s/w:
  numerous tasks use variety of modeling & programming languages

- Our goal:
  head toward unified representation of spacecraft
  accommodate complexities of spacecraft domain
  maintain capacity for knowledge abstraction

Research Goal

Approach

To reach this goal, we introduce:

- Model-based Programming
  (a novel approach to designing embedded s/w systems)
- Reactive Model-based Programming Language
  (a language for encoding model-based programs)

Today’s objective:
show how M-B Programming & RMPL provide a framework for robust sequencing
Model-based Programming

Example Scenario

Orbital Insertion Scenario:
Consider a simplified spacecraft, consisting of two identical redundant engines and a science camera.

Control Program

Control program specifies state trajectories:
- Must fire one of the two engines
- Set both engines to 'standby'
- Prior to firing engine, camera must be turned off to avoid plume contamination
- In case of primary engine failure, fire backup engine instead

Hidden State

- Note that states like (EngineA = Standby) are not DIRECTLY observable or controllable...

Given observations...
and command history...
can infer "hidden state"?

Similarly, can use system knowledge to "figure out" how to achieve this state.

Thinking in terms of such "hidden states" makes the task of writing the control program much easier.
Model-based Programming provides a way to infer and control these hidden states.

Simplified S/C System Model

Component modes...
described by constraints on variables...
deterministic and probabilistic transitions
Model-based Programming

Fundamental principle:
Control programs can be written by asserting and checking STATES which may be "hidden", rather than operating directly on observable or control variables...

Such a control program is input to a sequencing engine, for onboard execution.

An underlying mode estimation and reactive planning layer uses a model of the system to deduce the system state from the observations, and to figure out how to achieve a specified goal state.

RMPL Constructs
- constraint $c$
- concurrency (parallel $exp_1 \exp_2 \ldots$)
- sequential ordering (sequence $exp_1 \exp_2 \ldots$)
- conditional branching (if-thennext-elsenext $c \ then-exp \ else-exp$
- guarded transition (unless-thennext $c \exp$)
- iteration (always $exp$)
- extended guarded transition (when-done$\_c \exp$)
- iterated guarded transition (whenever-done$\_c \exp$)
- preemption (do-watching $c \exp$)

RMPL Model of Computation
- To support efficient execution, RMPL code is compiled into Hierarchical Constraint Automata (HCA):
OrbitInsert()::

\[
\text{(do-watching (EngineA = Firing) OR (EngineB = Firing))}
\]

\[
\text{(parallel (EngineA = Standby) (EngineB = Standby) (Camera = OFF))}
\]

\[
\text{(when-done (EngineA = Failing) AND (EngineB = Standby) AND (Camera = OFF))}
\]

\[
\text{(when-done (EngineA = Failing) AND (EngineB = Standby) AND (Camera = OFF))}
\]

\[
\text{(when-done (EngineA = Failing) AND (EngineB = Standby) AND (Camera = OFF))}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]

\[
\text{(EngineA = Firing) (EngineB = Firing) (Camera = OFF)}
\]
Compiling RMPL to HCA

OrbitInsert()::

[Diagram of OrbitInsert()]

LEGEND:

- (do-watching (EngineA = Firing) OR (EngineA = Standby))
- (EngineA = Standby)
- (EngineA = Standby) (Camera = Off)
- (EngineA = Failed)

EAS

EAF

EAR

EBS

EBF

EBR

EAREAS AND CO

Chart: 25

Chart: 26

Chart: 27

Compiling RMPL to HCA

OrbitInsert()::

[Diagram of OrbitInsert()]

LEGEND:

- (do-watching (EngineA = Firing) OR (EngineA = Standby))
- (EngineA = Standby)
- (EngineA = Standby) (Camera = Off)
- (EngineA = Failed)

EAS

EAF

EAR

EBS

EBF

EBR

EAREAS AND CO

Chart: 25

Chart: 26

Chart: 27

Executing HCA

Nominal (i.e. fault-free) orbital insertion scenario

initialize HCA by marking all start locations

Page 5
Executing HCA - Step 1

• initialize HCA by marking all start locations
• assert states from currently marked locations
• obtain state update
• take enabled transitions:
  • location’s state assignment achieved
  • transition and maintenance conditions currently hold true
  • mark new set of locations

Executing HCA - Step 2

• (EngineA = Standby) & (EngineB = Standby) achieved in this step
• two execution threads terminated & two transitions enabled

Executing HCA - Step 2

• initialize HCA by marking all start locations
• assert states from currently marked locations
• obtain state update
• take enabled transitions:
  • location’s state assignment achieved
  • transition and maintenance conditions currently hold true

Executing HCA - Step 1

• initialize HCA by marking all start locations
• assert states from currently marked locations
• obtain state update
• take enabled transitions:
  • location’s state assignment achieved
  • transition and maintenance conditions currently hold true

Executing HCA - Step 1

• initialize HCA by marking all start locations
• assert states from currently marked locations
**Executing HCA - Step 3**

- (EngineA = Firing) asserted in this step, but not yet achieved

**Executing HCA - Step 4**

- (EngineA = Firing) achieved in this step
- maintenance condition violated, HCA block exited

**Model-based Programming**

Advantages over traditional approaches to embedded s/w development:

- **Abstraction**: straightforward conversion of system engineering knowledge into flight code
- **Easier to specify desired state than control actions needed to reach it**
- **Powerful inference engines**: e.g. Livingstone (part of DS-1 Remote Agent), Burton
- **More flexible and robust than traditional rule-based engines**

**Modularity**

- model-based flight s/w can accommodate late design changes
- allows for transparent upgrading of deductive engines

**Model reusability**

- over time, build up database of models for subsystems and components
- reduce need for single-use flight code

**Verifiability**

- state-based control code & system models “readable” by system engineers

**Conclusion**

- We have discussed design of M-B Executive, consisting of:
  - sequencing layer coded in RMPL & compiled down to HCA
  - underlying deductive layer providing ME & RP capabilities, based on system models expressed in RMPL
- In current implementation, sequencing and deductive layer are distinct
- Eventual goals:
  - integration of both capabilities into a unified system, eliminating need for separately maintaining control program and system models
  - incorporate planning and scheduling capabilities (‘Kirk’ planner, currently under development)
  - accommodate continuous dynamics (Hybrid MPL, currently under development)

**RMPL Overview**

- Object-oriented language allowing a domain to be structured through a component or process hierarchy
- RMPL control programs can be viewed as deterministic state transition systems, acting on the plant by asserting and checking constraints in propositional state logic
- Propositions are assignments of state variables to values within their domains
- Reactive combinators allow flexibility in expression of complex system behavior and dynamic relations
- Similar to constructs in Timed CC (Saraswat, et al.)
Control program must capture following types of behavior:
• conditional branching
• iteration
• preemption
• concurrency

To serve as foundation for model-based execution, RMPL must provide key features of:
  – synchronous programming languages used in industrial embedded reactive systems
e.g. Esterel, Lustre, Signal
  – advanced robotic execution languages provide robust sequencing for ground-based robots and autonomous s/c
e.g. ESL, RAPs, TDL