Model-based Autonomy

Michel Ingham
Jet Propulsion Laboratory
California Institute of Technology

Paul Robertson
Model-based Embedded Robotic Systems Group
MIT Computer Science and Artificial Intelligence Laboratory

Sunday, October 9th, 2005

Who are we?

• Dr. Michel Ingham (JPL)
• Dr. Paul Robertson (MIT)
• Acknowledgment: Prof. Brian Williams (MIT)
Who are you?

• How about briefly sharing:
  – Your name
  – Your affiliation
  – What you do
  – The reason for your interest in this topic

• This way, we can try to tailor some of the discussion to your interests…

Logistics

• 8:00am to 11:50am
• 15 minute break around 9:50am
• Feel free to interrupt with questions at any time!
Outline

• Introduction & Overview
  – Motivation
  – Illustrative Scenario
  – Model-based Autonomy Architecture
• Model-based Programming
• Execution of Model-based Programs
• Fundamentals of Model-based Reasoning
• Modeling via State Analysis
• Advanced Methods
• Conclusion
Introduction & Overview

Vast Networks of Complex Embedded Systems

- We are creating vast networks of embedded systems that perform critical functions over long periods of time, often in harsh and uncertain environments.
- These long-lived systems achieve their increasingly ambitious goals by coordinating a complex network of devices.
- Spacecraft must achieve robustness by managing a complex set of subsystems, over a range of possible nominal and off-nominal scenarios.
- Programming these systems is becoming an increasingly daunting task.
We are creating vast networks of embedded systems that perform critical functions over long periods of time, often in harsh and uncertain environments.

These long-lived systems achieve their increasingly ambitious goals by coordinating a complex network of devices.

Spacecraft must achieve robustness by managing a complex set of subsystems, over a range of possible nominal and off-nominal scenarios.

Programming these systems is becoming an increasingly daunting task.
Vast Networks of Complex Embedded Systems

- We are creating vast networks of embedded systems that perform critical functions over long periods of time, often in harsh and uncertain environments.
- These long-lived systems achieve their increasingly ambitious goals by coordinating a complex network of devices.
- Spacecraft must achieve robustness by managing a complex set of subsystems, over a range of possible nominal and off-nominal scenarios.
- Programming these systems is becoming an increasingly daunting task.

Mission Sequencing: State of the Practice

- Time-tagged nominal command sequences

<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
<th>Parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>96-355/03:42:00.000</td>
<td>GS, SITURN, 490UA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:47:00.000</td>
<td>CMD, 7GYON, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:47:02.000</td>
<td>CMD, 7MODE, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:48:30.000</td>
<td>CMD, 6SVPM, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:50:32.000</td>
<td>CMD, 7SAFE, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:52:00.000</td>
<td>CMD, 6ASSAN, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96-355/03:56:08.000</td>
<td>CMD, 7VECT, 490UA412AA, BOTH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mission Sequencing: State of the Practice

- Time-tagged nominal command sequences
- If absolutely necessary, conditional behavior via rule-based monitors or hard-coded state machines

5.1.2.6 If the supply current for OXIDIZER TANK PRIMARY HEATER, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF OXIDIZER TANK PRIMARY HEATER.
5.1.2.7 If the supply current for HELIUM TANK PRIMARY HEATER, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF HELIUM TANK PRIMARY HEATER.
5.1.2.8 If the supply current for STAR TRACKER A, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF STAR TRACKER A.
5.1.2.9 If the supply current for STAR TRACKER B, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF STAR TRACKER B.
5.1.2.10 If the supply current for IMU PPSM-A, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF IMU PPSM-A.
5.1.2.11 If the supply current for IMU PPSM-B, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF IMU PPSM-B.
5.1.2.12 If the supply current for REACTION WHEEL 1, multiplied by the bus voltage, is greater than TBD watts, then the FPP shall issue a command to turn OFF REACTION WHEEL 1. In addition, the corresponding rules that monitor power dissipation of the remaining three reaction wheels shall be disabled. REACTION WHEEL 1 shall be flagged as “unavailable” to the G&C task in the MP.

Mission Sequencing: State of the Practice

- Time-tagged nominal command sequences
- If absolutely necessary, conditional behavior via rule-based monitors or hard-coded state machines
- Usual off-nominal behavior response is “safe mode”:
  - costly ground ops
  - lost science opportunities
- For critical mission sequences:
  - Safing mechanism is disabled
  - Hard-coded fault protection via highly-specialized software modules:
    - ad-hoc
    - complex
    - expensive to generate and test
Large collections of devices must work in concert to achieve goals
• Devices indirectly observed and controlled.
• Must manage large levels of redundancy.
• Need quick, robust response to anomalies throughout life.

Symptoms:
• Engine temp sensor high
• LOX level low
• GN&C detects low thrust
• H2 level possibly low

Problem: Liquid hydrogen leak
Effect:
• LH2 used to cool engine
• Engine runs hot
• Consumes more LOX
The Complexity Challenge

“Houston, we have a problem...”

- Quintuple fault occurs (three shorts, tank-line and pressure jacket burst, panel flies off).
- Ground assembles novel repair.
- Swigert & Lovell work on Apollo 13 emergency rig lithium hydroxide unit.
- Mattingly works in ground simulator to identify novel sequence handling severe power limitations.

Autonomy software should embody the innovation exemplified in Apollo 13 and other missions.

The Complexity Challenge

Mars Polar Lander

Leading Diagnosis:
- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 40m.
- Begins polling leg monitors to determine touchdown.
- Latched noise spike read as touchdown.
- Engine shutdown at ~40m.

Model-based Programming:
Creation of embedded & robotic systems that manage interactions automatically, by reasoning from models of themselves and their environment.

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

- **Model-based Programming** languages elevate the task to storyboarding and modeling.
  - Engineers program their high-level intentions in terms of how they would like the state of the world to evolve.
  - Programmers describe the world (system + environment) using commonsense models of normal and faulty behavior.

- **Model-based Executives** implement these intentions by reasoning on the fly.
  - They continually hypothesize the likely states of the world, given what they observe.
  - They continually plan and execute actions in order to achieve the programmer's intentions.

- **Model-based Autonomy** is the discipline of applying Model-based Programming principles to the control of complex embedded systems.
  - These systems achieve unprecedented robustness (“fault-awareness”) by leveraging the capabilities of their Model-based Executives.
  - They automate onboard sequence execution by tightly integrating goal-driven commanding, fault detection, diagnosis and recovery.

---

**Model-based Programs Reason about State**

Embedded programs interact with the system's sensors/actuators:
- Read sensors
- Set actuators

Programmers must reason through interactions between state and sensors/actuators.

Model-based programs interact with the system's (hidden) state directly:
- Read state
- Set state

Model-based Executives automatically reason through interactions between states and sensors/actuators.
Orbital Insertion Sequence: State-based Specification

planetary approach
rotate to insertion orientation and hold attitude
primary and secondary engines to standby
turn camera off
perform insertion burn

Set both engines to “standby”:
A off
B off

EngineA  EngineB
Orbital Insertion Sequence: State-based Specification

Set both engines to “standby”:

- A standby
- B standby

Primary and secondary engines to standby

Turn camera off

Perform insertion burn

Orbital Insertion Sequence: State-based Specification

Turn science camera “off” to avoid contamination from engine plume:

On

Science Camera
Orbital Insertion Sequence: State-based Specification

Planetary approach
rotate to insertion orientation and hold attitude
primary and secondary engines to standby
turn camera off
perform insertion burn

Turn science camera “off” to avoid contamination from engine plume:

off
Science Camera

Once both engines are in “standby” and the camera is “off”, fire the primary engine:

A standby
B standby

EngineA EngineB
**Orbital Insertion Sequence:**

*State-based Specification*

- Planetary approach
- Primary and secondary engines to standby
- Rotate to insertion orientation and hold attitude
- Turn camera off
- Perform insertion burn

**Once both engines are in “standby” and the camera is “off”, fire the primary engine:**

- EngineA firing
- EngineB standby

**Orbital Insertion Sequence:**

*Off-Nominal States*

- Planetary approach
- Primary and secondary engines to standby
- Rotate to insertion orientation and hold attitude
- Turn camera off
- Perform insertion burn

**If primary engine fails, fire secondary engine instead:**

- EngineA standby
- EngineB standby
Orbital Insertion Sequence: Off-Nominal States

planetary approach
rotate to insertion orientation and hold attitude
primary and secondary engines to standby
turn camera off
perform insertion burn

If primary engine fails, fire secondary engine instead:

B standby
A failed

EngineA EngineB

If primary engine fails, fire secondary engine instead:

A failed
B firing

EngineA EngineB
Typical Spacecraft Execution Architecture

Command Sequence

Sequence Execution, Real-Time Behaviors, & Fault Protection

Observations

System Under Control

Commands

Typical Spacecraft Execution Architecture

Command Sequence

Sequence Execution

Fault Protection

Real Time Behaviors

Observations

System Under Control

Commands
Time-tagged sequences of low-level commands and “macros”…

… executed by a nominal sequencing engine…

… with fault protection software running in parallel, ready to “take over” from nominal sequence execution when a fault monitor is triggered.

… augmented with event-driven behaviors when necessary…

Real Time Behaviors

Fault Protection

Sequence Execution

System Under Control

Commands

Observations

Typical Spacecraft Execution Architecture

Limitations of the Typical Architecture

Fault Protection is often considered an “add-on” capability, adjunct to the nominal control system and developed late in the project lifecycle, despite the fact that its design can uncover problems with the nominal control design.

Sequence designers’ intent is not explicit in the sequence

System requirements and understanding of behavior are not always directly traceable to the flight software design.

Complex interactions between these elements make it difficult and costly to validate flight software, and to have confidence that it will work reliably and robustly.

The boundary between State Determination and State Control is sometimes blurred, with no explicit representation of “State” in the software.

Command Sequence

Real Time Behaviors

Fault Protection

Sequence Execution

Observations

Commands

Typical Spacecraft Execution Architecture

Limitations of the Typical Architecture
Desirable Architectural Features

Control Specification

- Simple state-based control specifications with explicit intent

Onboard Executive

- Fault-awareness (in-the-loop recoveries)
- Models that are writable/inspectable by systems engineers
- Automated reasoning through low-level plant interactions
- Separation of state determination from control, with an explicit notion of state at the boundary

Model-based Programs and Executives Provide These Features

Model-based Program

- Simple state-based control specifications with explicit intent

Model-based Executive

- Fault-awareness (in-the-loop recoveries)
- Models that are writable/inspectable by systems engineers
- Automated reasoning through low-level plant interactions
- Separation of state determination from control, with an explicit notion of state at the boundary
Model-based Executive

Model-based Program

Control Program

System Model

Observations

System Under Control

Commands

Systems engineers think in terms of state trajectories…
Control Program specifies state trajectories:
- fires one of two engines
- sets 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

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) )
  )
)

Model-based Program

Engineers reason about how to achieve state trajectories using models of system behavior.
System Model: Formal Descriptions of State Behavior

System Model describes behavior of each component:
- nominal and off-nominal behavior
- qualitative constraints
- probabilistic transitions
- costs/rewards

One state machine per component, operating concurrently.

Model-based Autonomy Architecture

Model-based Program
- Control Program
- System Model

Model-based Executive
- Control Sequencer
- Deductive Controller

Configuration goals
State estimates
System Under Control
Commands
Observations
The Control Program is compiled into an executable form.

The Control Sequencer is responsible for generating, in real time, the sequence of configuration state goals prescribed in the Control Program.

The Deductive Controller is responsible for estimating the most likely current state based on observations from the system, and issuing commands to achieve the configuration goals.

The System Model is compiled into a form suitable for reasoning.
Example: Model-based Executive

- States like *(EngineA = Firing)* are not necessarily DIRECTLY observable or controllable
- When the Control Sequencer issues the configuration goal *(EngineA = Firing)*, the Deductive Controller…

![Diagram of Oxidizer tank and Fuel tank with valves and arrows indicating mode estimation and reconfiguration processes.](image-url)

**Mode Estimation**
- Deduces that thrust is off, and the engine is healthy

**Mode Reconfiguration**
- Plans actions to open six valves and executes them, one at a time
- Determines valves on the backup engine that will achieve thrust, plans needed actions and executes them.
- Deduces that a valve failed - stuck closed
• Introduction to Model-based Programming:
  – Control Programs
  – System Models