MDS
State-Based Architecture

Bob Rasmussen
May 16, 2001
State-Based Architecture

MDS

Determine → State → Control

Report
A system comprises project assets in the context of some external environment that influences them.

The function of mission software is to monitor and control a system to meet operators' intents.

MDS manages all essential aspects of this function via state:

- Knowledge of the system, including its environment, is represented over time in state variables.
- The behavior of the system is represented by models of this state.
- Interaction with the system is achieved via modeled relationships between state and interface data (measurements and commands), as mediated by hardware proxies.
- Information is reported, stored, and transported as histories of state, measurements, and commands.
- Operators' intent, including flight rules and constraints, are expressed as goals on system states.
A High Level View

State Knowledge

Models

State Determination

State Control

Elaborate

Coordinate

State

Goals

Telecommand

Measurements

Actions

Hardware Proxies

Hardware

Sense

Act

Report

Telemetry

Estimate

Propagate

May 16, 2001
Types of State

- Dynamics
  - Attitude, position, gimbal angles, altitude, ...
- Environment
  - Ephemeris, light level, atmospheric profiles, terrain, ...
- Device status
  - Configuration, temperature, operating modes, failure modes, ...
- Parameters
  - Mass properties, scale factors, biases, alignments, noise levels, ...
- Resource usage and allocations
  - Power and energy, propellant, data storage, bandwidth, ...
- Data product collections
  - Science data, measurement sets, ...
- DM/DT Policies
  - Compression/deletion, transport priority, ...
- Externally controlled factors
  - DSN schedule and configuration, ...
State Determination
Making Sense of the World

- One can act only on one's knowledge of the system
  - Knowledge is **what** you know, **not how** you know it
  - Observations (e.g., measurements) are not knowledge
- **Estimators** find "good" explanations for observations, given a model of how things work
  - Knowledge may be **propagated** into the future, given models and plans
- All knowledge is uncertain
  - Judgment must be based both on what is known, and on how well it is known
- However, local consistency of knowledge **is** achievable
Operators express their intent in the form of **goals**
- Goals declare **what** should happen, **not how**
- Goals may be expressed at any level

High level goals are elaborated recursively into lower level goals
- **Elaboration** may be conditional, in order to react to present circumstances
- **Coordination** of activities is accomplished by **scheduling**
- Conflicts are resolved, with priority as final arbiter

Knowledge of all states is maintained, as required to achieve goals
- Knowledge is compared to goal constraints to test for compliance

Corrective action is applied, as required to achieve goals
- Alternate methods of **achievement** may be applied at any level
- Unachievable goals (and their elaborations) are dropped individually without sacrificing others

Supports fault tolerance, critical activities, in situ autonomy, opportunistic science, and more
State Timelines

- **State timelines** maintain the value or set of possible values (e.g., a range) of a state variable as a function of time.
- They capture both knowledge and intentions about state.
Value Histories

- A container mechanism supporting functions that produce values over time (state variable timelines, measurements, commands, ...)
- Encapsulate the interface to data management persistent storage and data transport
  - Stored and transported as data products
- Leverage the use of models to preserve continuous information using less storage space
- Can also simply store a set of discrete value instances
- Controlled by storage and transport policies

Entries are combined and compressed as they age and are eventually deleted.

New Entries

May 16, 2001
State and Time Constraints

- Control is exerted over the system by imposing ...
  - **Constraints on states**, which limit the range of a state variable
    - State is allowed flexibility within these bounds
  - **Constraints on time**, which limit the duration between two **time points**
    - Time points are variable points in time
    - These times are allowed flexibility, but again, with constraints

- A state constraint between two time points is called a **goal**
- A time constraint between two time points is a **temporal constraint**

- Goals and temporal constraints are expressions of **intent**

- Success in constraint achievement is an objective matter
  - Criteria are explicitly expressed in constraint evaluation code
  - Directly verifiable during test, since constraints are explicitly evaluated
Constraint Networks

- Goals and temporal constraints each connect a pair of time points

![Diagram of Constraint Networks](image)

- Time points are often shared (e.g., one beginning as another ends)

- A collection of connected goals and temporal constraints form a **constraint network**
Resolving Conflicts

Three goals on the same state

The constraint

The time interval

Goal 1

Goal 2

Goal 3

Timeline

Goals 1 and 2 overlap, so they're compatible, as is

Goal 3 is incompatible with Goal 2, but it can wait

Crosshatched areas are outside goal constraints

May 16, 2001
Timeline Execution

- Goals are accepted if successfully placed on the timeline for the goal state variable.
- Goals are frozen and acted upon when they appear on the timeline in the immediate future.
- Goals are acted upon by achievers assigned to each state variable.
- Elaborators monitor execution and adapt plans, as necessary.

... achieve the goals.

Now, ...

... given the present goals ...

... and given the present state, ...

... achieve the goals.

May 16, 2001
Putting It Together

- Elaborators, scheduling, ...
  - Goal/event-driven
  - Planning and constraint solving
  - Analogous to sequencing, mode and configuration control, fault responses

- Achievers, DM/DT, ...
  - Provide system behaviors
  - Managed via goals and temporal constraints
  - Fairly conventional real-time monitoring and control processes

May 16, 2001
Ground-Flight Coordination

- State knowledge in both places
  - Common representation
  - Coordinated, consolidated & maintained, as appropriate
- Information exchanged via state variable proxies

- Ground based state determination
  - Typically things like orbit determination, calibration, …
  - Up-linked as necessary (trajectories, parameters, …)
- Flight based state determination
  - Typically things like attitude determination, device states, faults, …
  - Down-linked as available (part of telemetry)

- Similar story for goals, measurement, science data, …
Systems Engineering

- Systems and software engineering need to complement one another
  - Systems engineering must define the system structure and behavior
  - Software must understand the system structure and guide its behavior

- **State Analysis** is a model-based process defined by MDS to aid systems and software engineering
  - State analysis prompts comparatively methodical and rigorous analyses of systems
  - MDS permits the uniform expression of systems engineering concepts in software architectural terms
  - Resulting products map directly onto the MDS architectural elements
  - Most MDS adaptation requirements can be defined by state analysis

- Collaboration and documentation are presently supported by a web-based tool (the MDS State Database Server)
Backbone Workflow & System Engineering

Sanford Krasner
System Engineering Objectives

- Backbone Increment Objectives -> Collaborating Objects -> Scenario -> Functional Requirements on Each Object -> Functional Requirements on Frameworks
- Increment Objectives -> Collaborating Objects -> Scenario -> Functional Requirements on Each Object -> More Functional Requirements on More Frameworks
  
  .
  .
  .
  .
Backbone Workflow

- See spreadsheet at:
- Backbone spreadsheet drives the following concurrently and incrementally:
  - EDL time order
  - State Variable Development
  - Data Management Development
  - Simulation & Test Environment Development
  - Component and Infrastructure Development
Backbone Themes

- Create and Extrapolate States
- Add measurements, estimators, fault detection from measurements
- Add uncertainty and noise
- Add controllers, commands and goals; add redundancy between commands and measurements
- Add redundant information
- Add closed-loop control
- Add 3D dynamics
## Create and Extrapolate States

<table>
<thead>
<tr>
<th>Increment</th>
<th>Simulation</th>
<th>Flight</th>
<th>Transport</th>
<th>Ground</th>
<th>Framework</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JPL，“Rock”</strong></td>
<td>Start with single vertical descent within specified beginning time frame.</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Test results and data from <strong>JPL,“Rock”</strong>.</td>
</tr>
<tr>
<td><strong>Update altitude from ground estimates</strong></td>
<td>Update vehicle altitude and account for altitude in ground profile data.</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Test results and data from <strong>JPL,“Rock”</strong>.</td>
</tr>
<tr>
<td><strong>Update position 3-dimensionally</strong></td>
<td>Update vehicle altitude and account for altitude in ground profile data.</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Test results and data from <strong>JPL,“Rock”</strong>.</td>
</tr>
<tr>
<td><strong>Add solid surface</strong></td>
<td>Modify altitude based on solid surface.</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Test results and data from <strong>JPL,“Rock”</strong>.</td>
</tr>
<tr>
<td><strong>Continue altitude above solid surface</strong></td>
<td>Add a derived state variable for vehicle altitude above solid surface.</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Same data as simulation</td>
<td>Test results and data from <strong>JPL,“Rock”</strong>.</td>
</tr>
</tbody>
</table>

05/11/01
Create & Extrapolate States, Measurements & Estimators

- 1  Get initialized in a free fall
- 2  Update altitude from ground estimates
- 3  Make position 3-dimensional
- 4  Add Mars surface
- 5  Estimate altitude above Mars surface
- 6  Add ideal accelerometer
- 7  Add atmospheric drag
### MISSION DATA SYSTEM (MDS)

**Add measurements, estimators, fault detection from measurements**

<table>
<thead>
<tr>
<th>Device</th>
<th>Add measurements, estimators, fault detection from measurements (as defined in the text)</th>
</tr>
</thead>
</table>

**Add atmospheric drag**

Add a drag model to the vehicle position estimate (as a variable)

- Add a drag model to the vehicle position estimate (as a variable)
- Add a drag model to the vehicle position estimate (as a variable)

**Add accelerometer fault detection**

- Add accelerometer fault detection (as a variable)
- Add accelerometer fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)

**Add estimator fault detection**

- Add estimator fault detection (as a variable)
- Add estimator fault detection (as a variable)

**Add sensor fault detection**

- Add sensor fault detection (as a variable)
- Add sensor fault detection (as a variable)
Backbone Themes

• Fault Detection
  – 8 Fail accelerometer silent
  – 9 Fail accelerometer flat-line
  – 10 Fail accelerometer frozen
• 11 Compress accelerometer status
• Add uncertainty
  – 12 Add vehicle mass parameter
  – 13 Add initial condition uncertainty
  – 14 Add simple accelerometer noise
  – 15 Add accelerometer sampling jitter
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Add a simple gain switch</td>
<td>Controller &amp; Goal Increments</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Add a simple gain switch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Add a simple gain switch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Add a simple gain switch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Add a simple gain switch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Add a simple gain switch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More Backbone Themes

- Goals, Controllers & Commands; redundancy of commands & measurements
  - 16 Add a simple pyro switch
  - 17 Add supersonic chute
  - 18 Fail the supersonic chute control switch
  - 19 Make chute deployment switch redundant
  - 20 Detect proper supersonic chute deployment point
• Add redundant commands
  - 21 Add supersonic chute and backshell separation
  - 22 Detect proper supersonic chute and backshell separation point
  - 23 Add subsonic chute
  - 24 Add heatshield separation
  - 25 Add subsonic chute separation
  - 26 Detect proper subsonic chute separation point

• Add redundant measurements
  - 27 Add deployment and separation indicators
  - 28 Improve altitude estimator
  - 29 Add ideal altimeter
  - 30 Estimate altitude and surface elevation from altimetry
Add closed-loop control

- 31 Add a descent engine
- 32 Detect proper time for descent engine firing
- 33 Add a descent engine cutoff
- 34 Add a descent engine controller
- 35 Add contact indicator
3D World

- 36 Add vehicle rotation
- 37 Add an ideal 3 axis IRU
- 38 Add a set of ideal thrusters
- 39 Add an attitude control law
- 40 Add thruster history compression
EDL time order

- 1-dimensional universe (straight down)
- Create position/altitude state variable
- Free-fall in vacuum
- Update altitude from ground uplink
- Add atmospheric drag
- Fire pyro to release backshell & supersonic chute - increase atmospheric drag
- Release supersonic chute; deploy subsonic chute
- Separate heatshield
- Separate subsonic chute
- Use Altimeter
- Use descent engine to control descent
- Shut-off engine on estimate or contact sensor.
State Variable Estimate and Control Development

- Create state variable with time extrapolation model
- Update model via uplink
- Add "relative states" - spacecraft state wrt ideal, real surface
- Add sensor (accelerometer), measurements, estimation
- Add sensor failure mode detection, invalidate measurements
- Add measurement noise, initial condition uncertainty
- Add actuator (pyro switch), controller, command, discrete state (separation)
- Add goals, elaboration, execution
- Uplink goals
- Add actuator redundancy
- Add redundant measurements (altimeter), change WRT topology
- Add delegation framework (device control following descent profile)
- Add redundant altitude sensor (contact indicator)
Data Management Development

- Initialization
- Simple transport
- State Estimate Transport
- Log Events
- Uplink and use products
- Create & Initialize from persistent store
- Transport measurements
- Compress data
Simulation & Test Environment Development

- Representations compatible w/ ground
- Virtual time
- Simple acceleration model
- Stop run under specified conditions (error)
- Device simulation and interface
- Simple drag model
- Device failure models
- Random initial conditions and noise
Ground System Development

- Uplink model update
- Post-real time plotting
- Co-plotting of Flight and Truth (Sim) data
- Integrate to Third-Party Software visualization tool
- Real-time display of estimate functions, w/ uncertainty
- Text display
- Alarm display
- Goal-net generation and uplink

05/11/01
Component and Infrastructure Development

- Single-threaded components
- Multiple time frames
- Multi-threaded execution, synchronous and asynchronous communication
- Cyclic, time-alarm and event-based execution
**Identify Objects in the Increment**

**ews1**: Command (via EWS) Scheduler to schedule PositionEstimator to run at the commanded rate (cycles per period) and phase (executionSequence) from start time to stop time. Estimator phase (executionSequence) shall be set so as to execute after AccelerometerHwProxy has executed.

**ews2**: Command (via EWS) Scheduler to schedule AccelerometerHwProxy to execute at the commanded rate (cycles per period) and phase (executionSequence) from start time to stop time. EWS calls are synchronized and Scheduler returns status of request!

- **rate**: the execution frequency of this entity (cycle per period)
- **phase**: an offset sequence that request where in the rate group the object/component will execute (Sim uses executionSequence)

**ews3**: Request to SIM (via EWS synchronous interface) for the latest Accelerometer data. SIM returns sample data.
mds accelerometer increment-6
sequence diagram (Ver-H)

1) Initialize Scheduler prior to T0
2) ews1, Command (via EWS) Scheduler to schedule
   Position Estimator to run at once per second from T0 to
distant future.
   Position shall be set so as to execute after
   AccelerometerHwProxy has executed.

3) ews2, Command (via EWS) Scheduler to schedule
   AccelerometerHwProxy to run at twice per second from T0+10
to distant future.

T0: simulation start time

<table>
<thead>
<tr>
<th>VClock</th>
<th>AccelerometerHwProxy</th>
<th>Scheduler</th>
<th>Position Estimator</th>
<th>Position Basis</th>
<th>StateVariable</th>
<th>SIM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Run Estimator at 1 Hz (ews1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Schedule()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Run AccelerometerHwProxy at 2Hz (ews2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Schedule()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0</td>
<td></td>
<td></td>
<td>run(RTDuration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>getNexMeasurementSince(t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>From T0 to T0+9. Each Estimator run will</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>request measurements from proxy and proxy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>will return no data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0+10</td>
<td></td>
<td></td>
<td>no data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>run(RTDuration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>update()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0+10.5</td>
<td></td>
<td></td>
<td>run(RTDuration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>update()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0+11</td>
<td></td>
<td></td>
<td>run(RTDuration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>update()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The behavior is to get measurements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>from proxy until no more measurements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>available for the time specified</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4) Estimator will run with the most recent evidence and will
   retrieve measurements not older than 2 seconds

5) At T0+10 the proxy will run for the first time and generate
   one measurement. Estimator will run after proxy and retrieve
   measurement (not shown in diagram).

6) Scheduler sends run; message to AccelerometerHwProxy

7) AccelerometerHwProxy request SIM to get data via EWS
   synchronous interface for the latest Accelerometer data (ews3).

   SIM returns Accelerometer sample data as follows:
   -3 integers representing delta velocity since last sample
   X = DeltaVx (m/sec)
   Y = DeltagVy (m/sec)
   Z = DeltaVz (m/sec)

   The device proxy automatically updates the measurement

8) Scheduler executes Position Estimator after proxy for
   concurrent intervals.
   Each getNextMeasurementSince(t) call to proxy automatically
   returns a measurement as follows:
   AccelerometerMeasurement:
   -timestamp (inserted by proxy)
   -duration (sample interval in seconds inserted by proxy)
   -3 integers representing delta velocity along x, y, z axes

9) Update Position state knowledge
Requirement Based on Increment Scenario

- Requirements to support Flight
  - Increment-6 shall have the same deployments as Increment-5.
  - Component Scheduler shall accept component commands via EWS synchronous interface that schedule components to run at specified Rate and Phase offset.
  - The Component Scheduler shall provide a status return consequent to a component schedule command.
  - The Component Scheduler shall invoke run methods on successfully scheduled components.
  - Flight deployment shall include and adapt an AccelerometerHwProxy from the DeviceHwProxy frameworks.
Summary

- System Process goes from simplified mission scenarios to:
  - Capabilities allocated to implementable "units of work"
  - Capabilities allocated to framework capabilities.
- Scenarios, capabilities, frameworks are relevant to real missions
- Frameworks are available for adaptations
- Reference adaptations are available for reference
- State Analysis process developed based on Backbone
State Analysis

Sanford Krasner
What is State Analysis?

- MDS Adaptation is based on building Mission Software Systems out of MDS "framework elements": states, goals, measurements, commands, estimators, controllers, etc.
- MDS Frameworks support these elements:
  - interconnections, notification, initialization, persistent storage, etc.
- MDS Adaptation primarily instantiating existing frameworks
  - Filling in adaptation specifics:
    - Estimation and control Algorithms
    - Which measurements are used
    - Which other states are needed, etc.
- State Analysis is: Filling in adaptation specifics, in the pattern imposed by the MDS frameworks.
- State analysis encourages reuse:
  - Of MDS frameworks
  - Of adaptations from project to project
  - Of adaptation types for multiple instances
  - Of adaptations between flight and ground
Simple State Analysis Process

- Identify a state: Spacecraft Position
- Identify goals: Sitting at Target Landing Site
- How do you estimate it:
  - Propagate initial trajectory
  - Incorporate accelerometer measurements
  - Incorporate predicted effects of thruster commands
  - Incorporate altimeter measurements
- What other goals do you need (elaboration):
  - Set up attitude for entry
  - Chutes deployed at altitude/velocity
    - Pyro(s) fired at altitude/velocity
  - Thrusters firing to control descent
    - Thruster cat. bed heaters warm
      - Cat. bed switches on
  - Altimeter producing measurements
  - etc.
- What other states are implied? Repeat until done

12/11/2000
<table>
<thead>
<tr>
<th>Name</th>
<th>Edit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS/EDL 03 (Early Mars familiarization)</td>
<td></td>
<td>This state analysis supports the MDS/EDL Reference Mission &amp; Reference Spacecraft Definition. It defines the current best understanding of what the Spacecraft will look like at the end of FY03.</td>
</tr>
<tr>
<td>Container</td>
<td>BETA-ROOT-0</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td><a href="mailto:robert.c.barry@ipl.nasa.gov">robert.c.barry@ipl.nasa.gov</a></td>
<td></td>
</tr>
<tr>
<td>Categories</td>
<td>Add</td>
<td>GNC</td>
</tr>
<tr>
<td>Groups</td>
<td>Add</td>
<td>Rover Bat A Temperature</td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>GNC S/C Basebody</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GNC Entries</td>
</tr>
<tr>
<td>Actuator Types</td>
<td>Add</td>
<td>Heater</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Series/Parallel Switch Combination</td>
</tr>
<tr>
<td>Actuator Instances</td>
<td>Add</td>
<td>POSZ Aligned Reaction Thruster2 (POSX-NEG)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POSZ Aligned Reaction Thruster1 (POSX-POSY)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDL/Rover Battery A 2nd Heater</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDL/Rover Battery A 1st Heater</td>
</tr>
<tr>
<td>Command Types</td>
<td>Add</td>
<td>Thruster fire</td>
</tr>
<tr>
<td>Controller Types</td>
<td>Add</td>
<td>Solid State Power Switch (X2000) Control</td>
</tr>
<tr>
<td>Controller Instances</td>
<td>Add</td>
<td>Temperature Controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDL/Rover Battery A Temperature Controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDL/Rover Battery B Temperature Controller</td>
</tr>
<tr>
<td>Estimator Types</td>
<td>Add</td>
<td>Device Power Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
</tr>
<tr>
<td>Estimator Instances</td>
<td>Add</td>
<td>EDL/Rover Battery A Heater 1 Power Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDL/Rover Battery A Bulk Temperature</td>
</tr>
<tr>
<td>Goal Types</td>
<td>Add</td>
<td>Configure Power to a Device, Select Final Control Switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maintain EDL/Rover Battery A Temperature</td>
</tr>
</tbody>
</table>

12/11/2000

SK - 5
<table>
<thead>
<tr>
<th>Controller Instances</th>
<th>Add</th>
<th>Edit</th>
<th>EDL/Rover Battery A Temperature Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>EDL/Rover Battery B Temperature Controller</td>
</tr>
<tr>
<td>Estimator Types</td>
<td>Add</td>
<td>Edit</td>
<td>Device Power Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Temperature</td>
</tr>
<tr>
<td>Estimator Instances</td>
<td>Add</td>
<td>Edit</td>
<td>EDL/Rover Battery A Heater 1 Power Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EDL/Rover Battery A Bulk Temperature</td>
</tr>
<tr>
<td>Goal Types</td>
<td>Add</td>
<td>Edit</td>
<td>Configure Power to a Device, Select Final Control Switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Maintain EDL/Rover Battery A Temperature</td>
</tr>
<tr>
<td>Measurement Types</td>
<td>Add</td>
<td>Edit</td>
<td>PRT Voltage Measurement</td>
</tr>
<tr>
<td>Sensor Types</td>
<td>Add</td>
<td>Edit</td>
<td>PRT Temperature Sensor</td>
</tr>
<tr>
<td>Sensor Instances</td>
<td>Add</td>
<td>Edit</td>
<td>EDL/Rover Battery A Temperature Sensor</td>
</tr>
<tr>
<td>State Function Types</td>
<td>Add</td>
<td>Edit</td>
<td>Voltage-Temperature Calibration Curve</td>
</tr>
<tr>
<td>State Variable Types</td>
<td>Add</td>
<td>Edit</td>
<td>Trial State Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Altitude (distance, metres)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Temperature, Deg K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Position (distance, meters)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attitude (quaternion, non-dimensional)</td>
</tr>
<tr>
<td>State Variable Instances</td>
<td>Add</td>
<td>Edit</td>
<td>Mechanical Basebody: Attitude</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mechanical Basebody: Location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mechanical Basebody: Altitude wrt Landing Site</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mechanical Basebody: Horizontal Velocity wrt Surface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EDL/Rover Battery A Temperature</td>
</tr>
<tr>
<td>State Value Types</td>
<td>Add</td>
<td>Edit</td>
<td>Single Precision Floating Point</td>
</tr>
<tr>
<td>Subgoal Networks</td>
<td>Add</td>
<td>Edit</td>
<td>EDL/Rover Battery A Heater 1 Power Configuration</td>
</tr>
<tr>
<td>View Types (Views)</td>
<td>Add</td>
<td>Edit</td>
<td>Velocity View</td>
</tr>
</tbody>
</table>

12/11/2000
## STATE VARIABLE TYPE:

<table>
<thead>
<tr>
<th>State Type Name</th>
<th>State Type Description</th>
<th>Basis/Derived</th>
<th>Derivation (Conditional: only if a derived state variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Initialization Process**
- Default State Type Storage Policy
- Default State Type Transport Policy
- Default Value History Initialization Process

**State Views**
- State View Name
- State View Description (includes range)
- Parameters
- Return Value Type (includes representation of uncertainty)

**State Variable Instance**
- State Variable Instance: State Variable Instances – [list of state instances of this type]

**State Variable Type**
- State Variable Type: Parent State Variable Types – [list of state variable types this state variable type is derived from/uses to perform derivation]

(Conditional: only if a derived state variable)

(For each State Variable Type define its role. Prompt use to enter role name.)

**State Variable Type**
- State Variable Type: Dependent State Variable Types – [list of derived state variable types that use this state variable type in their derivation]

**Goal Type**
- Goal Type: Applicable Goal Types – [list of types of goals that can be applied to this type of state variable]

**State Function Type**
- State Function Type: State Function Types – [list of types of state functions that will be used for this state variable type]

(For each state function type there is a specific return type which is defined in state function type and displayed here. Link decorated with return type)

**Command Type**
- Command Type: Command Types – [list of types of commands which affect this state variable type]

**Command Type**
- Command Type: Command Types – [list of types of commands whose effects model use this state variable type]

**Measurement Type**
- Measurement Type: Measurement Types – [list of types of measurements which measure this state variable type]

**Estimator Type**
- Estimator Type: Evidence Receivers – [list of estimator types which use this state variable type for estimation]

**Estimator Type**
- Estimator Type: Estimators – [list of estimator types that may compute estimates of this state variable type]

(Conditional: only if a basis state variable)

**Controller Type**
- Controller Type: Controllers – [list of controller types which use this state variable type for control]

**Hardware Proxy Type**
- Hardware Proxy Type: Hardware Proxy Types – [list of hardware proxies types which can be used to measure or command this state. Referenced in measurement model or effects model]

(Hardware Proxy Types are sublists under the controller and estimator links listed above. They show up automatically and are viewable only under each controller and estimator link.)
STATE VARIABLE INSTANCE:

Basis/Derived/Proxy

(Conditions: If State Variable Type not derived then select basis or proxy,
else if State Variable Type is derived then derived
else if State Variable Type not yet defined then select basis, derived or proxy)

State Variable Instance Name (link if proxy)
State Variable Instance Description
Supported Views (list from State Variable Type, check applicable views)
Policy Notes (how & when, including but not limited to the following)
  Default State Type Compression Methods
    Compression Method Name
    Compression Method Description
    Compression Activation Method

(Link) State Variable Instance: Proxy Instances -- [list of proxies to this state variable instance]
  Conditional: only if a basis state variable. Decorated with deployment.

(Link) State Variable Instance: Basis & Sibling Proxy State Variable Instances -- [list of basis and sibling proxy state variable instances which this instance is a proxy to]
  Conditional: only if a proxy state variable. Decorated with deployment.

(Link) State Variable Instance: Dependent State Variable Instances -- [list of derived state variable instances that use this state variable instance in their derivation]

(Link) State Variable Instance: Parent State Variable Instances -- [list of state variable instances this state variable instance is derived from; associated in some way (ordered as keyword) to derivation arguments]
  Conditional: only if a derived state variable.

(For each State Variable Instance associate its role from its State Variable Type)

(Link) Deployment Instance: Deployment Instance -- [deployment where this state variable instance lives]

(Link) State Variable Type: State Type -- [type of state variable this state variable instance is an instance of]
  Conditional: only if basis or derived.

(Link) Controller Instance: Controller -- [the controller instance that controls this state variable instance]

(Link) Controller Instance: Controllers That Use -- [list of controller instances this state instance supplies estimated values to]

(Link) Estimator Instance: Evidence Receivers -- [list of estimator instances which use estimated values of this state instance as evidence]

(Link) Estimator Instance: Estimator -- [estimator that computes estimates of this state variable instance]
  Conditional: only if a basis state variable.

(Hardware Proxy Instance: Hardware Proxy Instances -- [list of hardware proxies which can be used to measure or command this state. Referenced in measurement model or effects model]
(Hardware Proxy Instances are sublists under the controller and estimator links listed above. They show up automatically and are viewable only under each controller and estimator link.)
Goal Elaboration

- Each goal may have other (sub)goals that have to happen:
  - Before (preparation - warm cat. bed heaters for 90 minutes before using)
  - During (keep accelerometers powered on during descent)
  - After (safely shut-down engine after landing)
- Elaboration adds subgoals to support parent goal (and so on recursively)
- Working on Goal Elaboration Tool to support elaboration drawings, integrate to State Database
The flash indicates that a goal is typically used only as an event to signal a condition. Nevertheless, it may have an elaboration (e.g., to assure that detection is possible).

This is a goal to be elaborated. It is associated with two time points. There is an implied $[0, \infty)$ time constraint between these time points, but additional time constraints may be present elsewhere. Not all goals have parameters.

A dotted line between two time points is used to indicate that the two time points at either end are really the same time point.

Goals to be elaborated may be executable or non-executable.

Goals with no elaboration are called "terminal". This is indicated simply by showing nothing below this dashed line.

Elaborations can introduce new goals. These are called subgoals.

Elaborations can introduce new temporal constraints. They are shown by solid lines between two time points.

A temporal constraints specifies the minimum and maximum acceptable duration of the interval between two time points. The following shorthand is used:

Relative:
- $\text{Precedes} = [0, \infty)$
- $\text{Delay} \ D = [D, D]$
- $\text{At least} \ L = [L, \infty)$
- $\text{At most} \ M = [0, M]$

Absolute (relative to Epoch):
- $\text{At} \ T = [T, T]$
- $\text{After} \ T = [T, \infty)$
- $\text{Before} \ T = [0, T]$

Absolute temporal constraints are specified as relative temporal constraints with respect to the Epoch.

Delay $D = [D, D]$

Precedes $= [0, m]$

At least $L = [L, m]$

At most $M = [0, M]$

Absolute $T = [T, T]$

After $T = [T, \infty)$

Before $T = [0, T]$

Condition

Non-executable goals are shown as rounded rectangles (ovals are okay too).

Before July 4, 2076

This is also the end time point.

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]

Item Name:
State Name:
Subgoal Name[parameters]
Precondition invokes required initial conditions wrt altitude, attitude and velocity.

**Basebody: Six DOF**

State: In Terminal Descent Guidance

- **Thruster (Descent, All):** Force and Torque: Following Terminal Descent Guidance Control Law
- **Outriggers:** Condition: Is Deployed

Delay TBD (60s?)

- Can't start lower and meet still terminal guidance initial conditions
- **Basebody: Altitude wrt Landing Site:** Starts Above [TBD m]

Delay TBD (5s?)

- **Subsonic Chute:** Condition: Is Deployed
- **Subsonic Chute:** Condition: Is Being Separated

**Subsonic Chute:**

Schedule hazard detection early enough, but don't make proceeding dependent on success.

- **Landing Area:** Hazard-Free Locations: Known

Proceeds, even if horizontal velocity is still larger than desired.
Basebody: Horizontal Velocity wrt Landing Site: Is Being Damped

Thruster (Descent, All): Force and Torque: Following Horizontal Velocity wrt Landing Site Damping Control Law

Basebody: Horizontal Velocity wrt Landing Site: Is Known to [TBD m/s]