Mission Data System

Overview and Control Loop Features

Presentation to Control Systems Workshop
October 15-16, 2003

Slides by Robert Rasmussen, Daniel Dvorak, Kirk Reinholtz, Alex Moncada
Compiled by Matt Barry, Kenny Meyer
Copyright 2003, by the California Institute of Technology. ALL RIGHTS RESERVED. United States Government Sponsorship Acknowledged. Any commercial use must be negotiated with the Office of Technology Transfer at the California Institute of Technology.
Agenda

- MDS Overview
- Bridging the Gap with State Analysis
- MDS Control Loops
Overview
The application domain of MDS is: mission information, control, and operations ... of physical systems ... for unmanned space science missions

Scope includes flight, ground, and simulation/test software

Adopts a product line (multi-mission) approach that exploits commonalities across missions

MDS has three aspects:
- An information and control system architecture
- A systems engineering methodology
- Reusable and adaptable framework software
What MDS Provides

- **System engineering methodology**
  - Structured process for disciplined analysis
  - Emphasizes model-based design for estimation and control
  - Makes interactions explicit, exposes complexity

- **Architectural patterns**
  - State architecture
  - Component architecture

- **Frameworks and adapter’s guides**
  - Reusable building blocks in object-oriented design
  - Guides for how to adapt it for concrete tasks
  - Examples of framework usage

The State-Based Architecture

State and component architectures are pieces of the same design methodology

10/14/2003
What MDS is for

- Highly reusable core software for flight, ground, and test
- Synergistic systems & software engineering
- Reduced development time and cost
- Improved development processes
- Highly reliable operations
- Increased functionality

MDS Vision

A unified control architecture and methods for flight, ground, and test systems that enable missions requiring reliable, advanced software
Bridging the Gap with State Analysis

A unifying approach for

Systems Engineering and Software Engineering
There Is a Big Gap Between…

- What **Systems** Engineers do:
  - Define and analyze the capabilities a system must have
  - Establish the decomposition of functionality
  - Provide key algorithms for accomplishment of these functions
  - Integrate, test, operate, and maintain a system

- And what **Software** Engineers do:
  - Define the software architecture of a system
  - Provide the tools and techniques for software development
  - Design and build and test the software to provide the required functions
• Systems engineering is **outward** looking
  - Mission scenarios
  - Functional decomposition
  - System analysis
  - Performance requirements
  - Resource allocations
  - Command and telemetry dictionaries
  - Flight rules and constraints
  - Control laws
  - Failure modes analysis
  - Fault protection
  - Test procedures

• Software engineering is **inward** looking
  - Languages, libraries, operating systems…
  - Concurrent threads, processes, memory management…
  - Real time execution
  - Patterns, abstractions, general algorithms…
  - Data representation, serialization…
  - Interprocess communication
  - Deadlocks, access violations, exceptions…

... Different!
State Analysis:

• A uniform, methodical, and rigorous approach to...
  • Discovering, characterizing, representing, and documenting the **states** of a system
  • Modeling the **behavior** of states and **relationships** among them
  • Capturing the mission **objectives** in detailed **scenarios**
  • Keeping track of system **constraints** and operating **rules**
  • Describing the **methods** by which objectives will be achieved
  • Recording information about hardware **interfaces** and **operation**

• For each of these design aspects, there is a simple but strict **structure** within it is defined

• This structure is comprised of

  **Common Framework Elements**
A Taste of State Analysis

Standard Questions:

What do you want to achieve?
Move rover to rock

What's the state to be controlled?
Rover position relative to rock

What evidence is there for that state?
IMU, wheel rotations, sun sensor, stereo camera

What does the stereo camera measure?
Distance to terrain features, light level, camera power (ON/OFF), camera health

How do you raise the light level?
Wait until the sun is up

Where is sun relative to horizon?

Common Framework Elements:

Goal

State Variable

Measurements

Model

State Effects Model

Etc.

10/14/2003

MDS 12
Common elements appear in....

State-Based Architecture

MDS

Report

State

Determine

Control
Common elements appear in...

State Architecture (the details)

The color coding is meant to convey similarities, e.g., estimators and controllers are goal achievers, sensors and actuators are hardware adapters, measurements and commands are time-tagged items.

10/14/2003
Software Built to Requirements

Architecture designed for complex interactions demanded by "physics"

Common Elements

Systems and Software Engineering use same Language
- No translation, consistent representation
- Simplified inspection
- Simplified implementation
- Verifiable requirements

Reusable Frameworks

10/14/2003 MDS 15
State is Central

• 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.
State-based Goal-driven Architecture

- Estimators interpret measurement and command evidence to estimate state.
- State variables hold state values, including degree of uncertainty.
- A goal is a constraint on the value of a state variable over a time interval.
- Models express mission-specific relations among states, commands, and measurements.
- Controllers issue commands, striving to achieve goals.
- Hardware proxies provide access to hardware busses, devices, instruments.

Telecommand

State Knowledge

State Determination

Models

Hardware

Sensors

Actuators

Measurements

Actions

Options

Report
State Knowledge
Everything You Need to Know

- Dynamics
  - Vehicle position & attitude, gimbal angles, wheel rotation, ...
- Environment
  - Ephemeris, light level, atmospheric profiles, terrain, ...
- Device status
  - Configuration, temperature, operating modes, failure modes, ...
- Parameters
  - Mass properties, scale factors, biases, alignments, noise levels, ...
- Resources
  - Power & energy, propellant, data storage, bandwidth, ...
- Data product collections
  - Science data, measurement sets, ...
- DM/DT Policies
  - Compression/deletion, transport priority, ...
- Externally controlled factors
  - Space link schedule & configuration, ...

... and so on
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 and other evidence, 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, one can **achieve** local consistency of knowledge
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 reactivity, opportunistic science, and more
Hardware Proxies
Connecting With the World

- Provide local software representatives of system hardware
  - Delineating the abstract model of the system (*including time!*)
  - Translating raw input/output data into abstract declarations about state
    - **Measurement models** relate incoming data to state
    - **Command models** do the same for outgoing data

- Augments system hardware with supplemental behaviors
  - Sampling
  - Time and metadata tagging
  - Data format translation
  - Local tight control loops
  - Data compression
  - I/O sequencing and synchronization
  - Data buffering and routing
  - Error checking
  - Data preprocessing
  - Etc.

- Isolates state frameworks from platform specific interfaces
  - Built on ACE middleware
  - Real, simulated, or abstract hardware
  - Real or virtual time
State Analysis Procedure

- Models suggest how states should be estimated
  - Estimators often use models directly
  - You may identify multiple ways to know a state, depending on circumstances and need

- Estimators are “goal achievers”

Example: Analyzing State Determination
Monitoring the System and Its Own Actions to Determine State

- Define a State Variable
- How should state knowledge be updated?
- How well must the state be known?
- Define a Knowledge Goal
- How will this be achieved?
- Model an Effect
- Define an Estimator
- Model a Command
- Model a Measurement
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 intent about state.
Constraint Networks

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

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

- A collection of connected goals and temporal constraints form a goal network.
Resolving Conflicts

- Example: three goals on the same state

The constraint

Goal 1

Goal 2

Goal 3

The time interval

flexible start

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

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

Executable Goal Timeline

Crosshatched areas are outside goal constraints

10/14/2003
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 $\textit{achievers}$ assigned to each state variable
- Elaborators monitor execution and adapt plans, as necessary

... and given the present state, ...  
... achieve the goals.

Intent  
Knowledge  
Time

Now, ...
- 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
What are the primitive elements?

- **State Variables**, **Estimators**, **Controllers**, and **Hardware Adapters** are the fundamental elements that make-up the state-based software architecture of MDS; aka MDS Diamond

- All control loops use variations of this MDS diamond pattern
MDS separates state determination from state control, coupled only through state variables (Architectural Theme)

- Frequently when estimation and control are entangled the state information is never made explicit
  - The SV wouldn’t not exist
  - Users of needed information run the risk of having multiple interpretations for the same data
- For consistency, simplicity, clarity, and testability separate state determination logic from control logic
State Knowledge Cont.

- Estimators makes use of (the inputs)
  - Device evidence, such as
    - Sensor Measurements
    - Commands issued by hardware adapters to H/W
    - Models
  - Other state variables

- Estimators keep state information up to date (the output)
  - Updates state knowledge by using SV's state function
Parachute Status Estimator Example

Parachute Status SV Estimator

- Drag Coefficients consistent with Parachute is DEPLOYED
- Pyro Switch is ProbablyClosed OR CLOSED
- Pyro Switch is ISOPEN OR Failed_Open
- Drag Coefficients NOT consistent with Parachute is DEPLOYED

Unknown

STOWED

DEPLOYED

Probable Deployed

Changes to:

- PyroSwitch is ISOPEN OR
- Pyro Switch is Failed_Open OR
- Drag Coefficients NOT consistent with Parachute is DEPLOYED

Pyro Switch is CLOSED

Drag Coefficients consistent with Parachute is DEPLOYED

Gaining Knowledge uncertainty rather than to quantify it

10/14/2003

MDS 33
Measurements, Models, and their use

Control Loop

- State Variables
- Estimators
- Controllers
Hardware devices such as sensors provide raw information.

This raw information is processed by local software interfaces that represent system hardware called Hardware Adapters (HA).

HA are the only elements that interface with system hardware and process raw sensor data.

One HA for every required software interface fidelity:
- For example, one for physical h/w and one for each simulation fidelity.
- Isolates the controlling system from platform specific interfaces.
Example of Measurements, Models, and their use

- **Parachute Deploy Pyro Switch Status Measurement**

  - The switch measurements shall be represented as follows:
    - Deployment time stamp
    - Switch position
    - Health

  - Measurement Constituents:
    - Pyro switch measurement is as follows:
      1. Deployment time stamp (Ephemeris Time Frame)
      2. 2-bit Integer representing switch position;
         0 = OPEN, 1 = CLOSED, 2 = Failed

  Sample pseudo code for switch measurement model:

  ```
  def PyroSwitchStatus-switchstatus):
    if health == 'Unhealthy':
      return (Enum.Failed)
    Deployment TimeStamp == Ephemeris Time Frame:
    SwitchPosition = 2-bit Integer
    position = 0 for OPEN, 1 for CLOSED, 2 for Failed
    PyroSwitch measurement is consistent with switch measurement model for position CLOSED
    Parachute is Deployed
    OR PyroSwitch measurement is not consistent with switch measurement model for position CLOSED
    with persistence
    Parachute is Deployed
    OR PyroSwitch status is OPEN
    OR PyroSwitch status is CLOSED
    with persistence
  ```

10/14/2003

MDS 36
Example of Measurements, Models, and their use Cont.
Alternate approaches to Estimator design

- Table

<table>
<thead>
<tr>
<th>Parachute Status State</th>
<th>Pyro Switch Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stowed</td>
<td>ISOPEN</td>
</tr>
<tr>
<td>Probably Deployed</td>
<td>Probably Closed</td>
</tr>
<tr>
<td>Deployed</td>
<td>Failed Open</td>
</tr>
</tbody>
</table>

- Hypothesis testing algorithm

Estimator will distinguish between the different operating and failure modes

If \( F_{PyroSw}(CLOSED) \) equals Measurement Sw Position and
Parachute Status is (Deployed or ProbablyDeployed)
return \( CLOSED \)

If \( F_{PyroSw}(ISOPEN) \) equals Measurement Sw Position and
Parachute Status is Stowed
return \( ISOPEN \)

If \( F_{PyroSw}(ISOPEN) \) equals Measurement Sw Position and
Parachute Status is (ProbablyDeployed)
return \( Probably Closed \)

If \( F_{PyroSw}(CLOSED) \) not equal to Measurement Sw Position and
Parachute Status is (Deployed)
return \( Failed Open \)

The point here is that algorithm design is business as usual

10/14/2003
MDS 37
State Knowledge Patterns

- Distillation Pattern (Estimator to Estimator Pattern)
  - Estimation is staged
  - The output products are distilled "measurements" dependant on fewer states
  - Later estimation stages take advantage of state information already extracted

- For example, Terrain Map State Variable

Camera measurements are functions of camera model

Stage-1 estimator processes evidence (meas. and SV's)

Distilled measurement not dependent on camera model

Stage-2 estimator updates state
State Control

Control Loop

State Variables

Estimators

Hardware Adapters
State Control Cont.

- Control is defined as closing the loop through State
  - Architectural Theme; State determination is considered separately from State Control

- State control is in the business of getting what you want

- Controllers are the achievers of state control

- Similar properties to estimators in that they are both Achievers, however

- Controllers know how to control a state (not determine it)
State Control Cont.

- They meet the objectives given to them regarding the state of the system under control
  - Controls what they know through state

- Controller design can be modal (state machines) or whatever makes sense
  - Describes and captures the required behavior
  - Adaptation specific

- Controlling algorithms
  - Can be simple or complicated such as Terrain Hazard Avoidance
  - Driven by need and performance
• Real-time execution
  • Control of state can be periodic or event/data driven
    • For example a Wheel Motor Controller runs periodically at 4 Hz and has also subscribed to Wheel Position SV and Wheel Motor Health SV.
    • Under nominal conditions (motor “is healthy”) wheel motor controller executes cyclic
    • However if at any time the motor becomes “unhealthy”, controller will run on motor health state notification and safe the motor.

• Controllers are the only MDS elements that can control state

• Controllers are the only MDS elements that can issue commands to hardware
• Controllers are responsible for achieving a requested state constraint

• Controllers are goal achievers because they work to satisfy a constraint on the value of a state variable

• Intent is specified through **Goal** (Constraint on State)
  - For example “Deployed Parachute” or “Pyro Switch is Closed”

• Controllers are told “what” to do, they determine what “actions” to take

{
  "The What"
  
  **Goal**

  "The How"
  
  Achieve Goals by closing loop through **State**

  "The Action"
  
  Commands to hardware as needed
Commands, Models, and their use
Hardware devices such as actuators (and some sensors) require actions in order to initiate their state transition.

We call these actions Commands.
- For example "Closing Pyro Switch", "Opening Thruster Latch Valve"

These commands are processed by local software interfaces that represent system hardware called Hardware Adapters (HA).

HA are the only elements that interface to all system hardware; sensors and actuators.
Example of Command, Models, and use

- Parachute Deploy Pyro Switch Command and Model

- This command changes the position of the pyro switch
- Command Constituents:
  - Switch Position. Range: Close, Open

Example pseudo code for switch command model

```plaintext
F_PyroSw(pyro Switch state, command)
```

Controller Algorithm:
Can achieve closing the switch as long as the switch state is not Failed_Open
If switch state is Closed and constraint is to Close the switch then do nothing; 'idle'
If switch is isOpen or Unknown and constraint is to Close the switch then issue command to Close switch
If switch state is Failed_Open and constraint is to Close the switch then report goal failure and no nothing; 'idle'

10/14/2003
State Control Patterns

- Controller to Hardware Adapter Pattern
  - This is the normal control pattern
  - A state variable can only be controlled by one controller
  - Controllers can control multiple state variables
  - Only Controllers can send commands to hardware adapters

- Delegation of Control Pattern
  - Control is staged and coordinated
    - Controllers delegate authority to other Achievers
    - Delegation nesting can be as deep as needed
    - All coordination done via Goals

10/14/2003
State Control Patterns Cont.

- Two delegation sub-patterns

- **Controller to Controller** coordination
  - For example; Need to control spacecraft attitude with thrusters

- **Estimator to Controller** coordination
  - For example; Need to park rover and update traversing terrain

See Session-6 for more details on delegation and goals.
**Command Sequence vs. Goal Network**

**Command Sequence**
- Specifies commands to be executed at specific times
- Multiple sequences can run concurrently
- Original operator intent not expressed in sequence
- Sequence planning depends on good predictions of state
- Fault protection is designed independently

*(Open-loop control)*

**Goal Network**
- Specifies goals to be achieved within time windows
- All timelines run concurrently
- Operator intent apparent with traceability from high-level to low-level goals
- Timing of goal begin/end can depend on state as well as time
- Reaction to goal failure is all part of the design

*(Closed-loop control)*
Conclusion Cont.
Conclusion Cont.

- Control is defined as closing the loop through State, with State Determination considered separately from State Control.

- Estimators and Controllers are the Achievers of state knowledge and control.

- State knowledge and State Control are specified through **Goals**.

- Patterns help coordinate state control and knowledge.
MDS Team (a partial list)

- MDS Program Office Manager ................. Allan Sacks
- MDS Project Manager.......................... John Lai
- Chief Architect ................................... Robert Rasmussen
- Chief Programmer ............................... Kirk Reinholtz
- Systems Engineering Lead...................... Sandy Krasner
- Build Manager ................................... George Rinker
- Mission Planning & Execution Lead .......... Tom Starbird
- Data Management/Data Transport Lead ...... David Wagner
- Simulation & Test Lead ......................... Mohammad Shahabuddin
- Component Architecture Lead ................ Nicolas Rouquette
- Software Engineer .............................. Kenny Meyer
- Deputy Architect .............................. Dan Dvorak