Practical Application of Model-based Programming and State-based Architecture to Space Missions

Gregory Horvath, Michel Ingham
Jet Propulsion Laboratory, California Institute of Technology

Seung Chung, Oliver Martin, Brian Williams
Computer Science and Artificial Intelligence Laboratory, MIT

Space Mission Challenges for Information Technology 2006

July 17 – 20, 2006

Pasadena CA

Overview

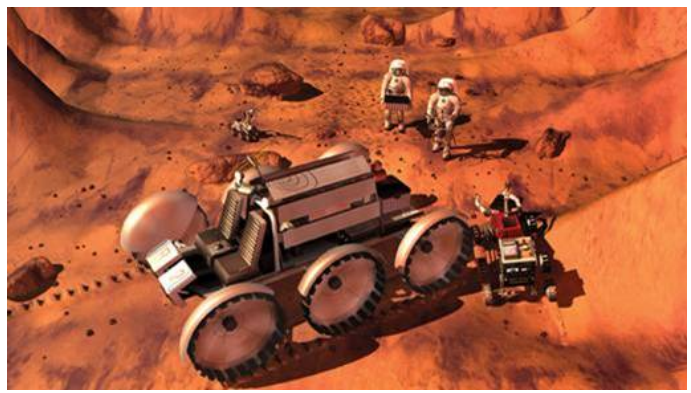
- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Overview

- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Motivation

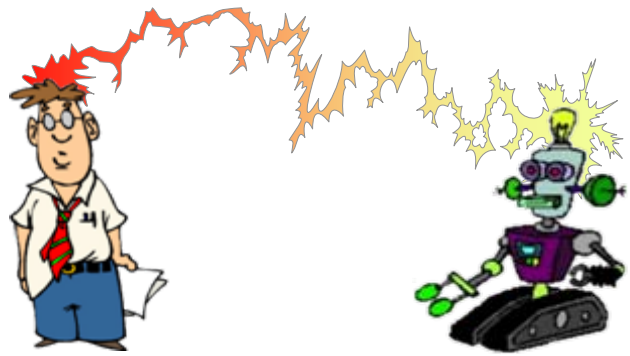
- Spacecraft operate in a harsh, *uncertain environment*.
- Spacecraft achieve robustness by managing a *complex set of redundant subsystems*, over a range of possible *nominal and off-nominal* scenarios.
- Reasoning about interactions within and between subsystems during sensing and actuation is complex and error-prone, and requires significant *interaction between systems engineers and software programmers*.
- The traditional gap between systems and software engineering can lead to errors that manifest themselves as *software defects*.
- NASA software engineering practice has been challenged to create and verify space systems that assure ***correctness, reliability, and robustness*** in a ***timely and cost effective*** manner.



Objective/Vision

Current Practice: System Software is a Surrogate for Systems Engineers, and Software Engineers perform the transformation.

Our Goal: Models from Systems Engineers are provided directly to the embedded system, which is capable of reasoning through them to accomplish mission objectives and manage the health of the system!



Approach

- The State-of-the-Art is still far from fully achieving this Vision
- Yet even incremental steps toward this Vision can greatly improve on the current practice, producing software that is:
  - Less error-prone,
  - More robust to off-nominal situations,
  - Cheaper,
  - Easier to verify, etc…
- Three facets of our approach:
  - Develop **representations** and **algorithms** that endow the embedded system with the requisite reasoning capabilities *(Model-based Programming and Execution)*
  - Integrate these technologies into a **principled software architecture** that facilitates adaptation to a particular application *(State-based control architecture and MDS software framework)*
  - Provide Systems Engineers with **methods** and **tools** that help them reason through the system design and develop models in a *rigorous* way *(State Analysis)*

Overview

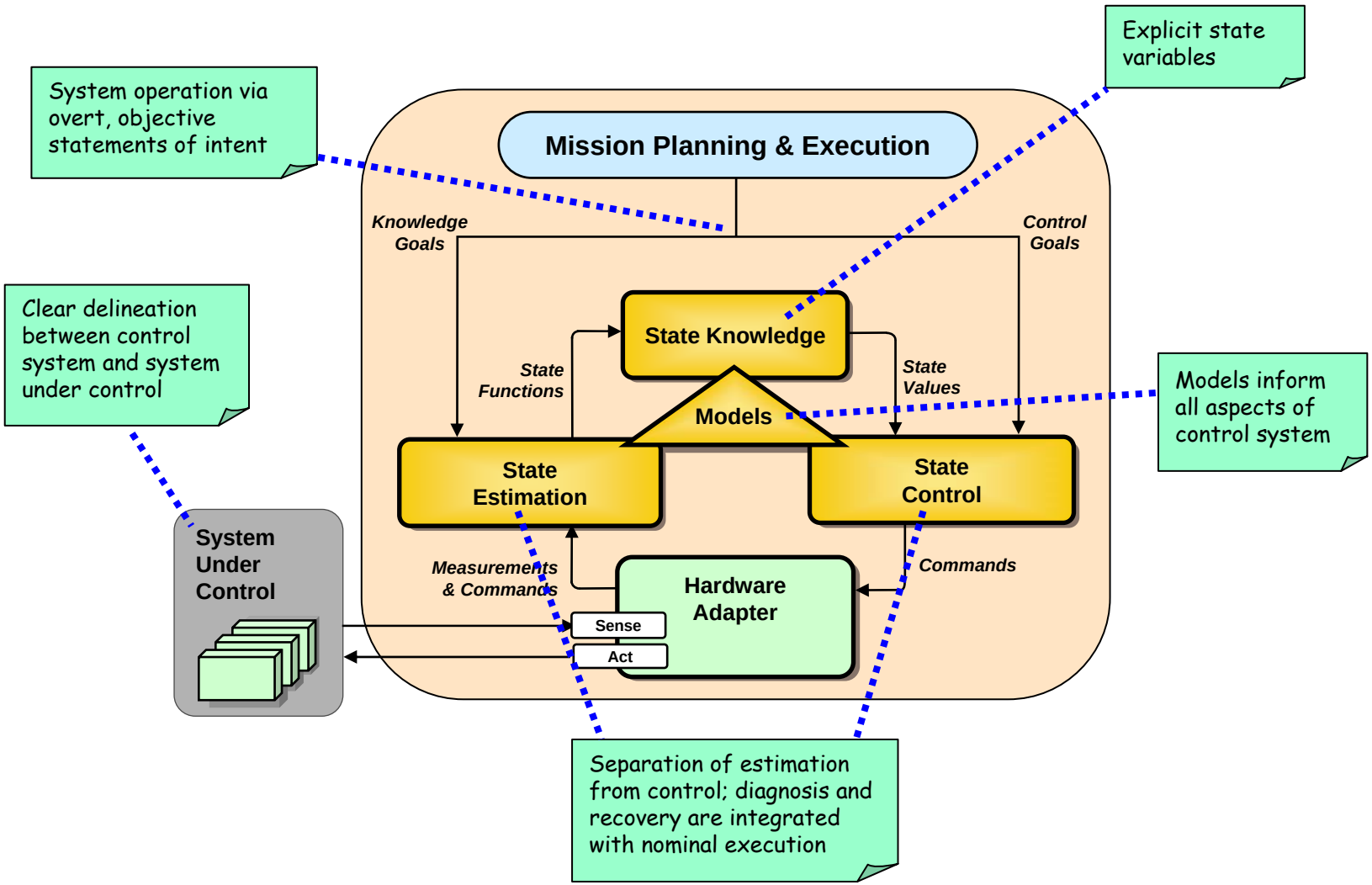
- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Background: The Mission Data System

MDS is a set of technologies and methodologies, including:

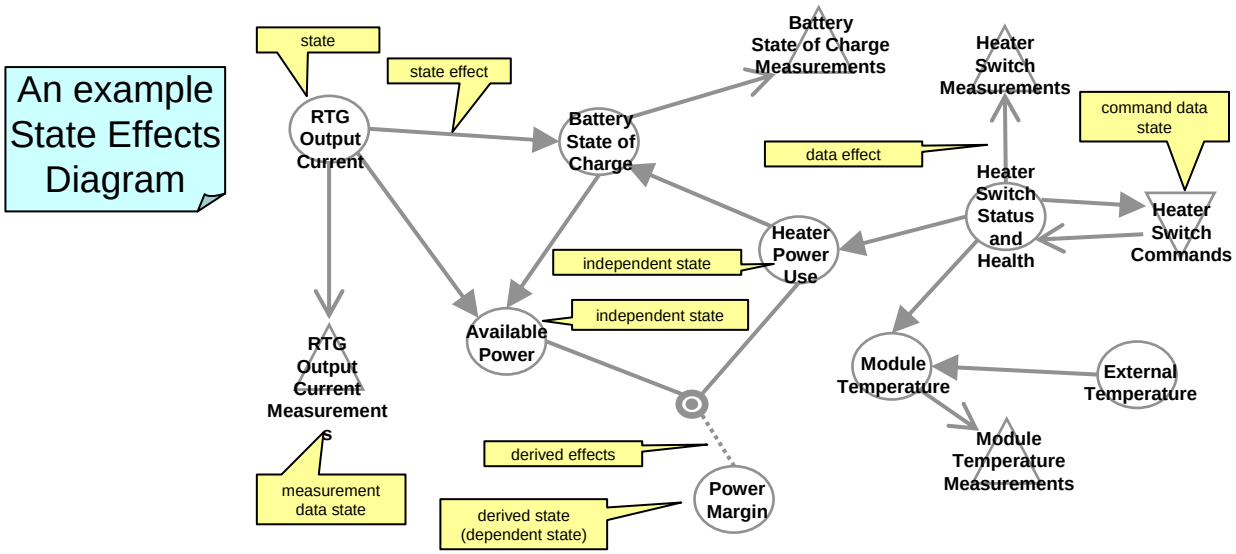
- A State-Based Architecture for Control Systems
- A Systems Engineering Process for Control Systems ("State Analysis")
- A Software Framework for Control Systems (a state-of-the-art embedded software framework based on a modular component architecture implemented in C++ and supported on multiple operating systems)

Background: State-based Control Architecture



Background: State Analysis

- A uniform, methodical, and rigorous approach for…
  - Discovering, characterizing, representing, and documenting the **states** of a system
  - Modeling the **behavior** of states and **relationships** among them, including information about hardware **interfaces** and **operation**



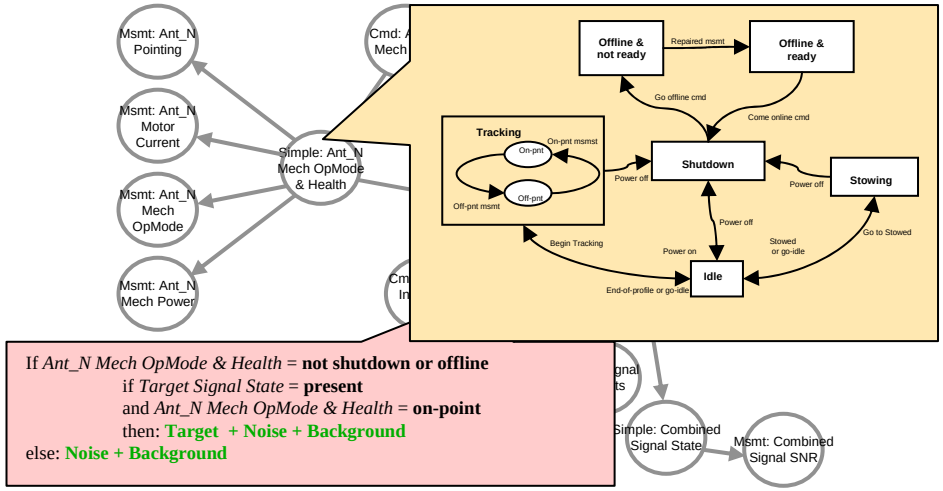
  - Capturing the mission **objectives** in detailed **scenarios** motivated by operator **intent**
  - Keeping track of system **constraints** and operating **rules**, and
  - Describing the **methods** by which objectives will be **achieved**

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

Overview of State Analysis

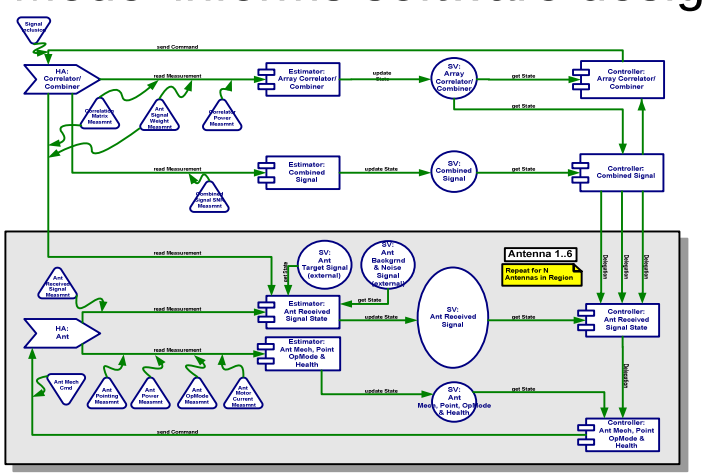
1. System to be controlled

2. State Analysis produces model



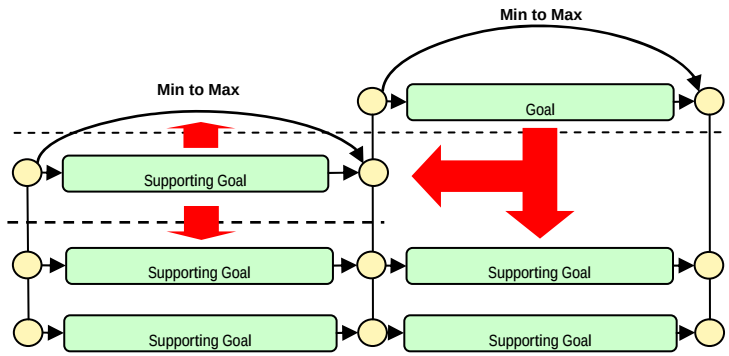
If *Ant_N Mech OpMode & Health* = **not shutdown or offline**
if *Target Signal State* = **present**
and *Ant_N Mech OpMode & Health* = **on-point**
then: **Target + Noise + Background**
else: **Noise + Background**

3. Model informs software design

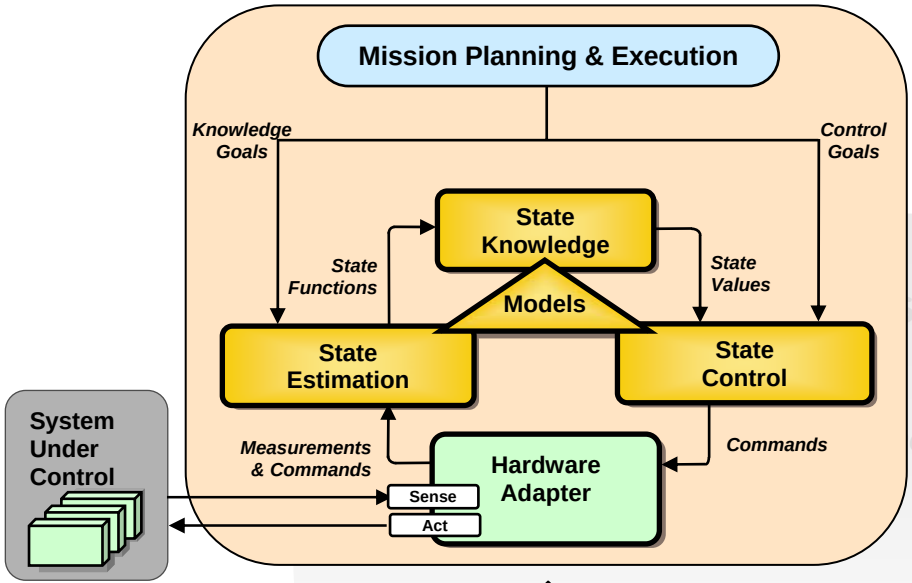


DSAN Collaboration Diagram

4. Model informs operations

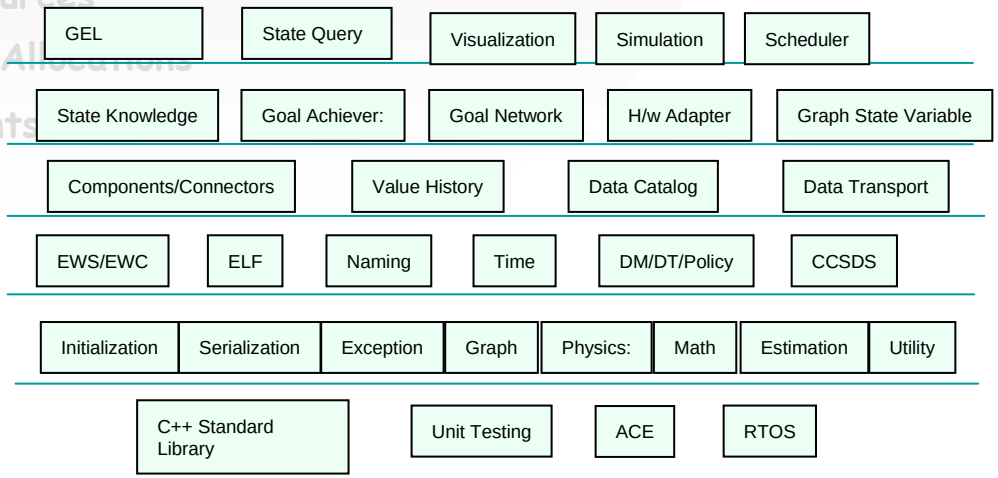


Background: MDS Software Frameworks



State Analysis:
Model-based Requirements Map Directly to Software

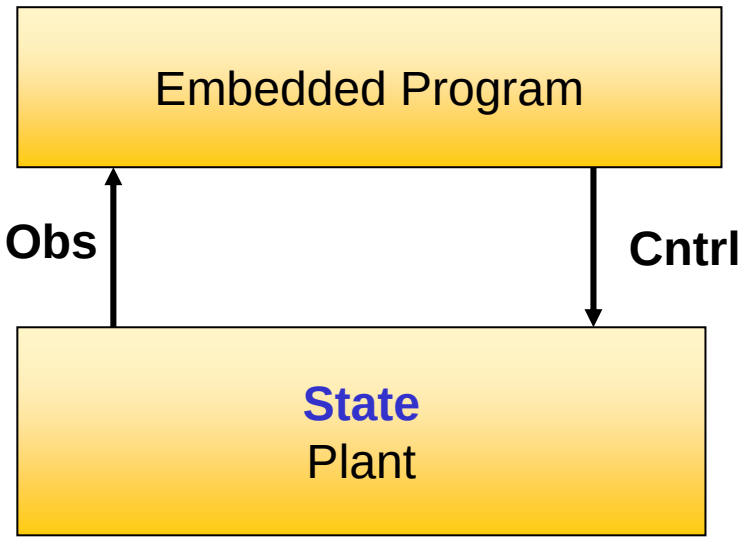
Common Vocabulary
Reduces Errors of Translation



Background: Model-based Programming

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

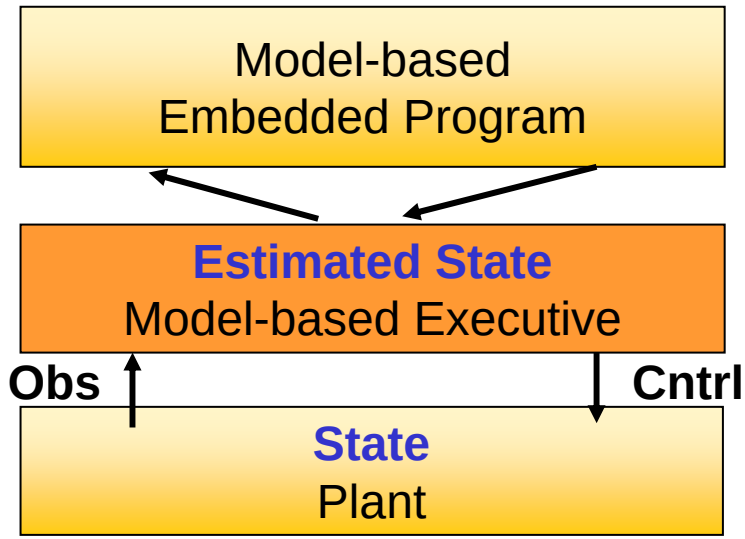
- Read sensors
- Set actuators



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

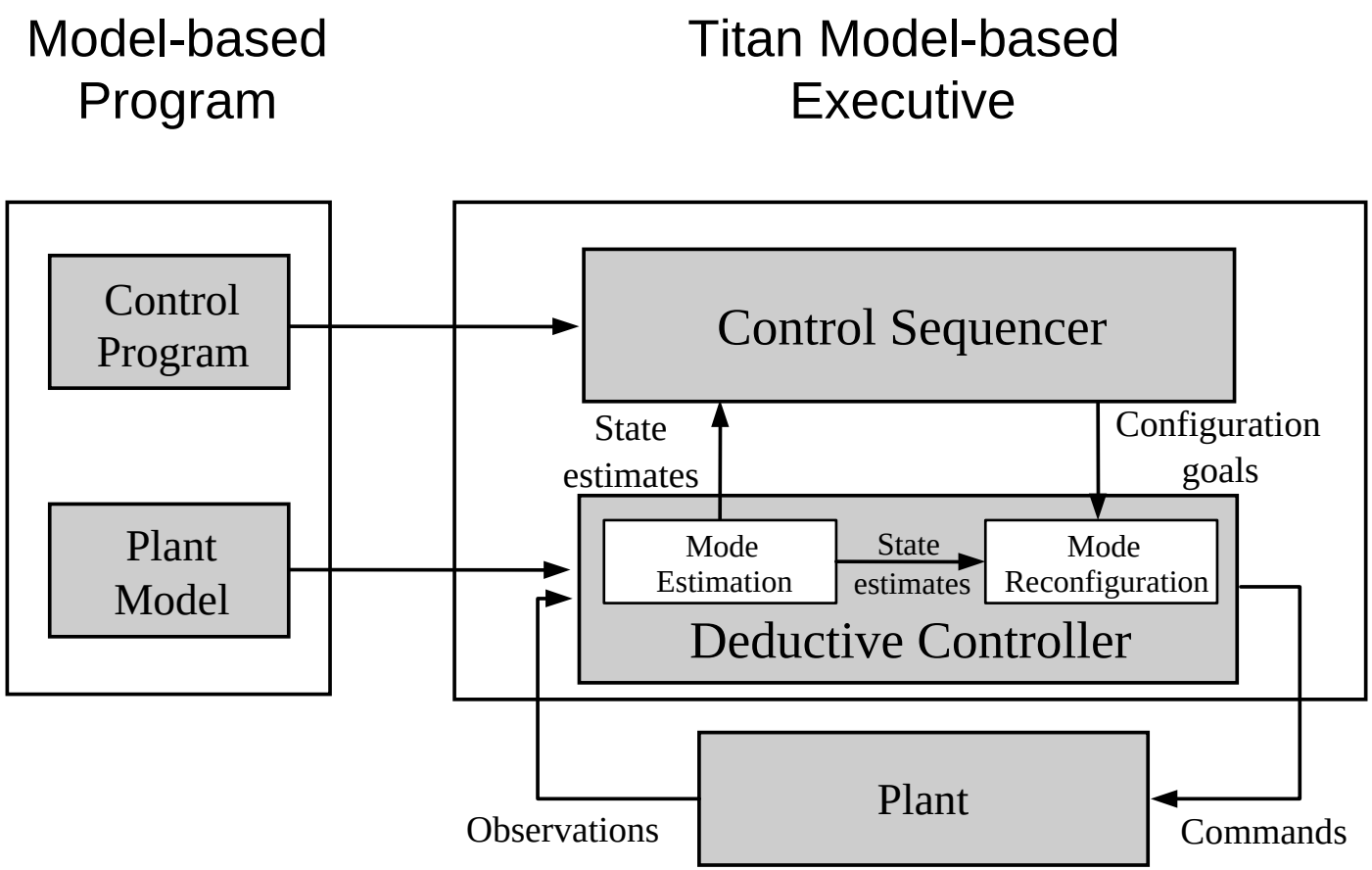


Model-based Executives automatically reason through interactions between states and sensors/actuators.

Background: Titan Model-based Executive

- Developed at MIT
- Two-tiered architecture, consisting of
  - A ***control sequencer*** which executes the model-based control program, and
  - A ***deductive controller*** which
    - Reads sensors and generates state estimates, (*Mode Estimation*)
    - Uses the state estimates and input from the control sequencer to generate commands to be sent to the hardware elements (*Mode Reconfiguration*)

Model-based Execution Architecture



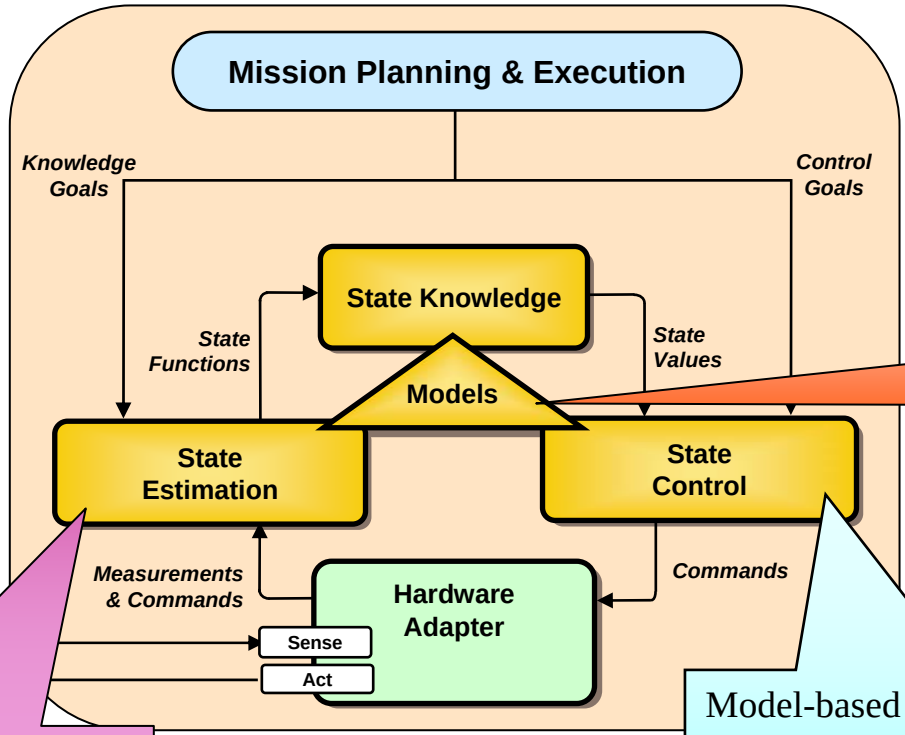
Overview

- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Compatibility Analysis of MDS and Titan Architectures

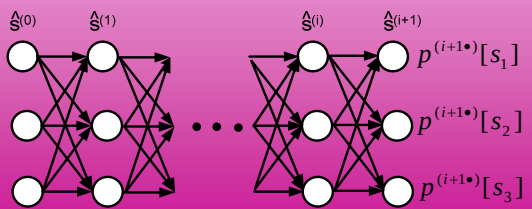
Titan Concept	MDS / State Analysis Concept
Models represented as factored POMDPs	No prescribed representation, but Titan representation compatible with Stats Analysis requirements
System state represented as discrete assignment to each state variable in the system	Continuous assignment to a set of MDS State Variables
Mode Estimation / Mode Reconfiguration	Estimator / Controller
Configuration Goal *(Ex. – Transition to and Maintain State X)*	Goals, Elaborations (*Ex. – Transition to State X; Maintain State X)*

Integrating Model-based Programming and Execution into the Architecture

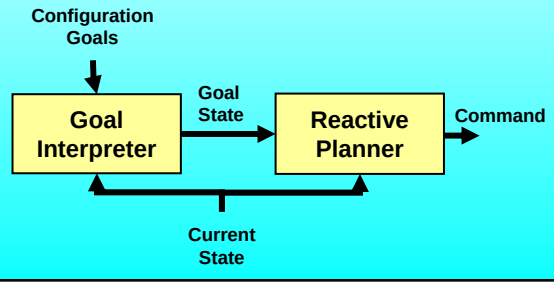


Models are used EXPLICITLY by the control system

Model-based reasoning algorithm:
- Tracks a limited set of most-likely states
- Explores state space in best-first order



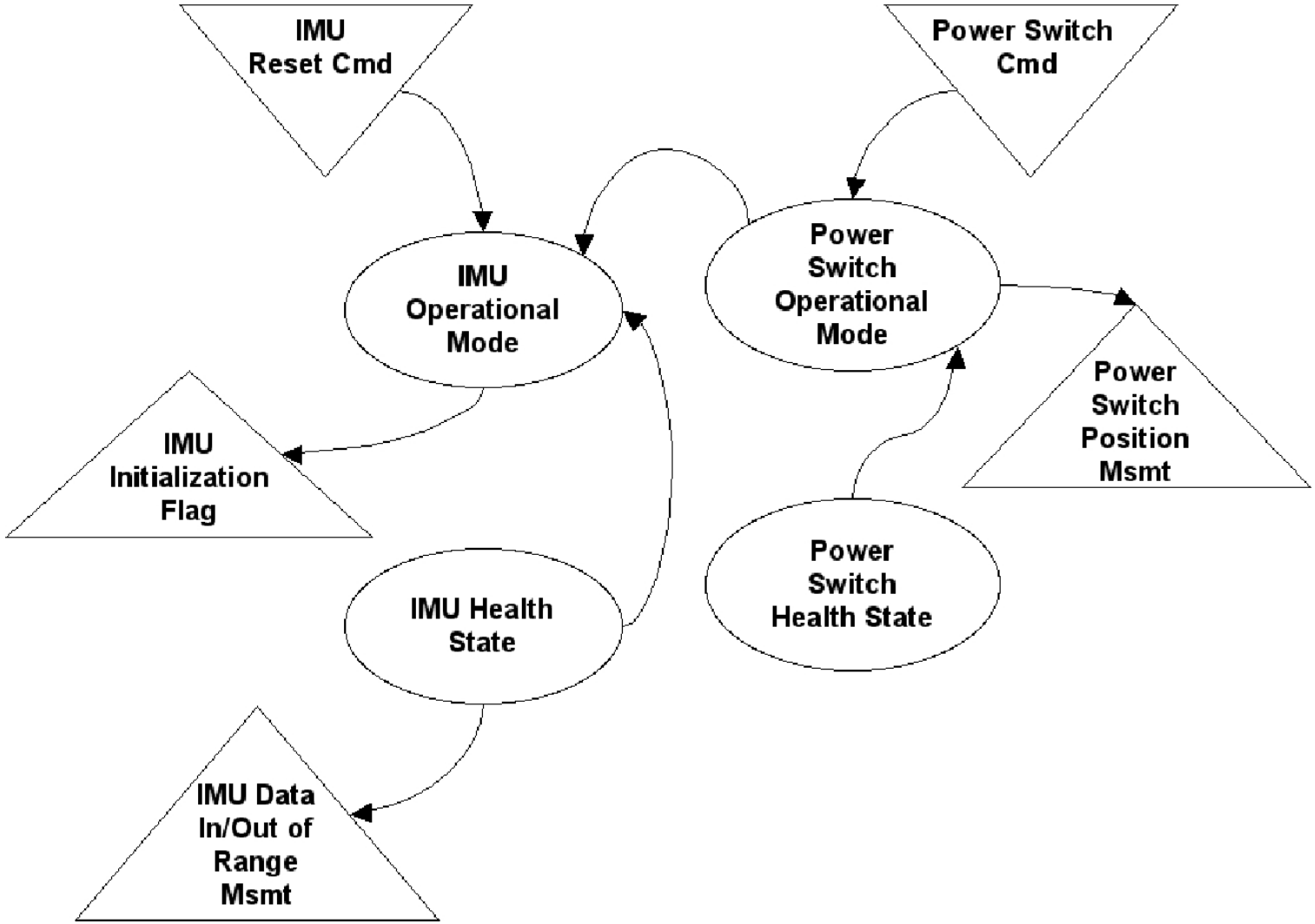
Model-based reasoning algorithms to compute a series of commands that progress the system towards a least-cost state that achieves the goal.



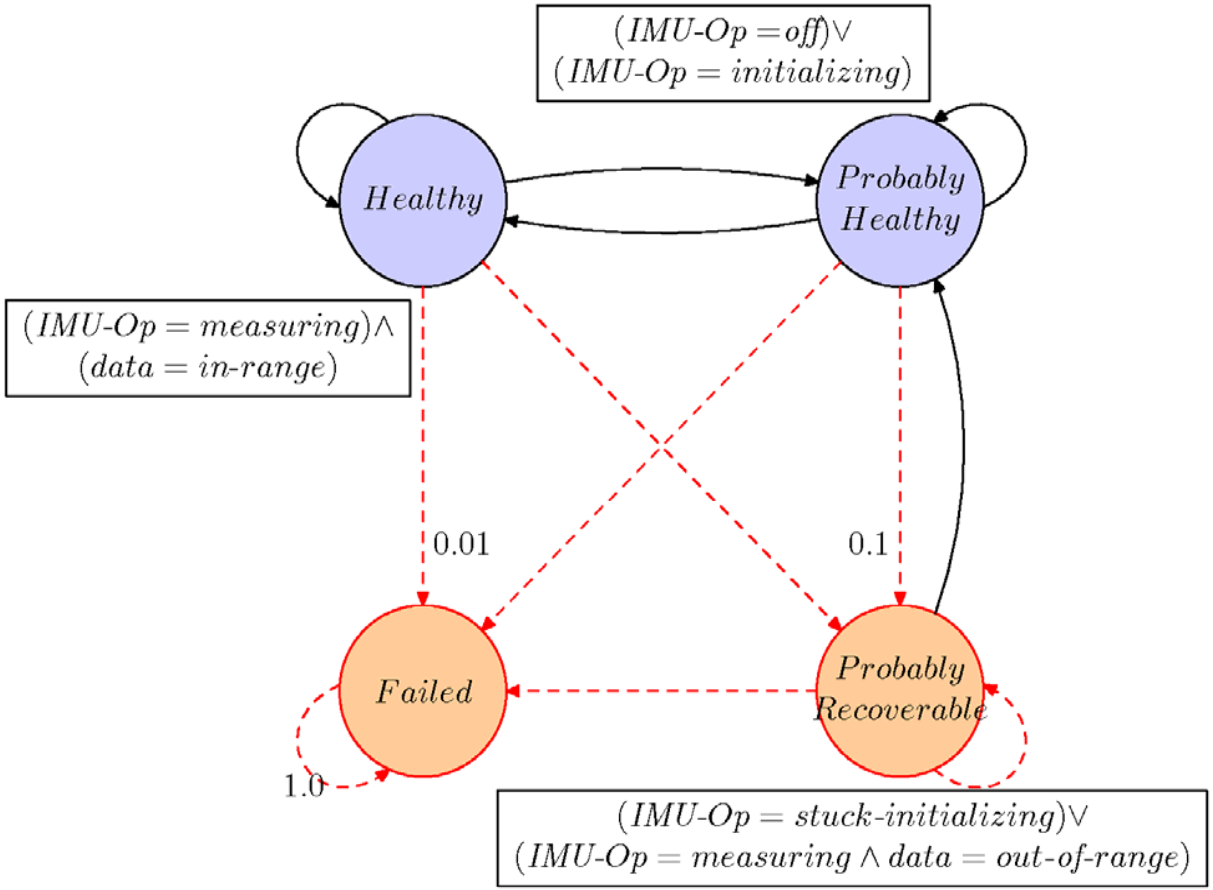
State Analysis and Modeling

- IMU subsystem plus associated power switch chosen as demonstration subsystem
  - Leveraged modeling work performed in support of a previous MDS Mars Lander prototype demonstration
- Modifications to existing products were made to support Titan integration
  - State Effects Diagram modified to separate State Variables which were previously combined (IMU OpMode & Health, Power Switch OpMode & Health)
  - Software collaboration diagram modified to reflect consolidation of estimation and control functionality for all State Variables
  - Existing estimation and control algorithms no longer necessary (replaced by Titan ME and MR functionality)
- Modeling activities focused on developing Titan subsystem models (factored POMDPs)

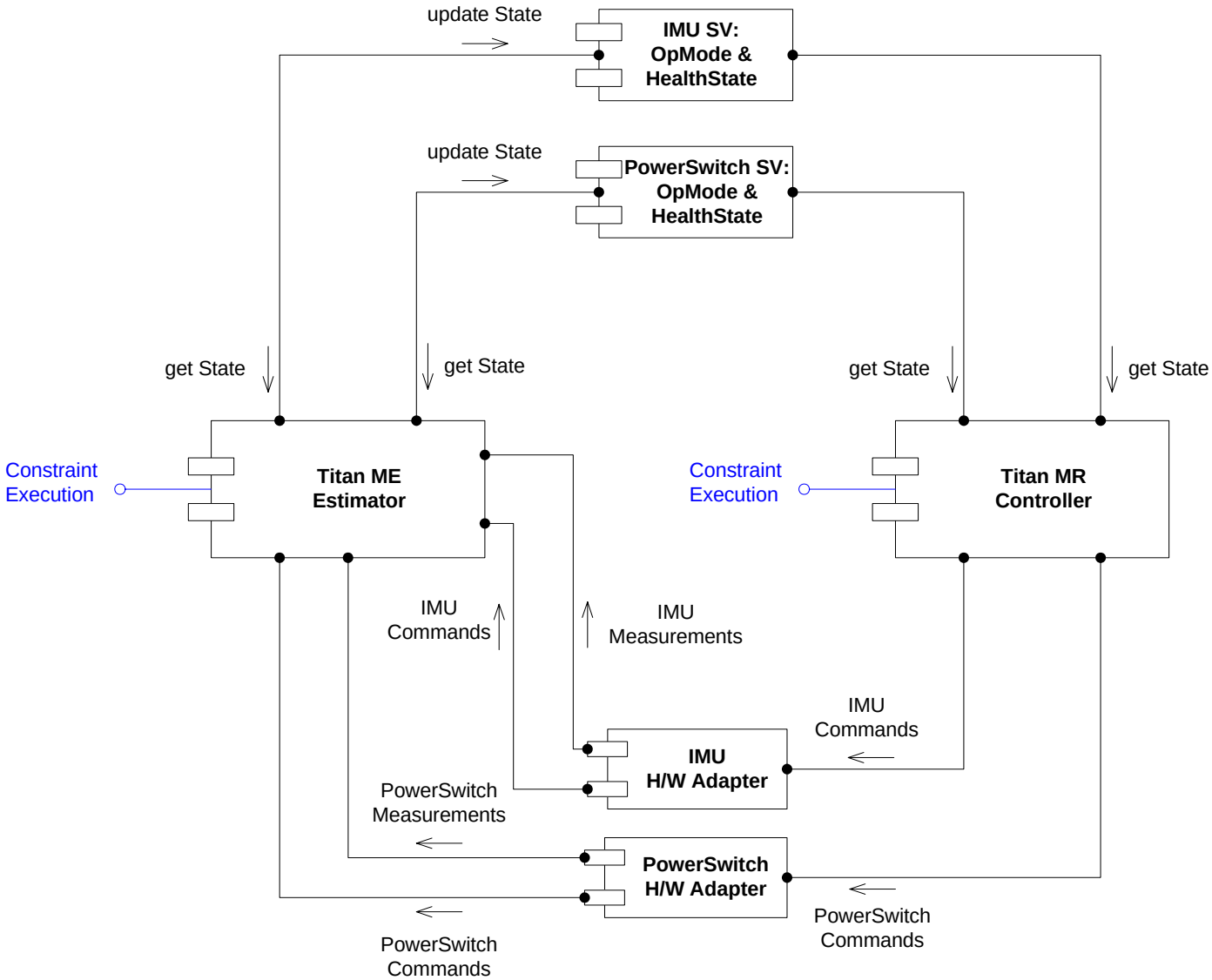
IMU Subsystem State Effects Diagram



Titan Subsystem Model: IMU Health

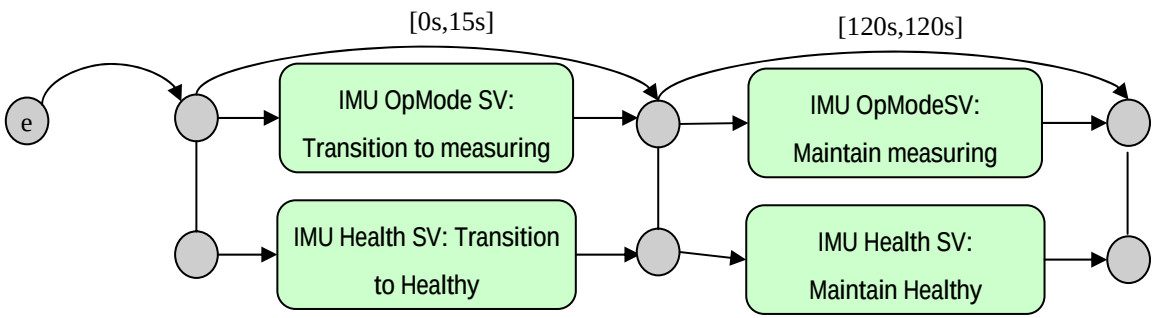


Integrating Model-based Programming and Execution into the Software



Testing Program

- Two main thrusts of the testing program
  - ***Software Verification***, focused on interfaces between MDS frameworks, newly-developed adaptation code, and the Titan engine.
  - ***Model Verification***, focused on uncovering any latent errors or incompatibilities in the source subsystem models developed for the task.
- All tests performed against a software simulation, allowing for testing of both nominal and off-nominal scenarios
- All tests based on the same operational scenario, and therefore used the same activity plan or *goal network.*

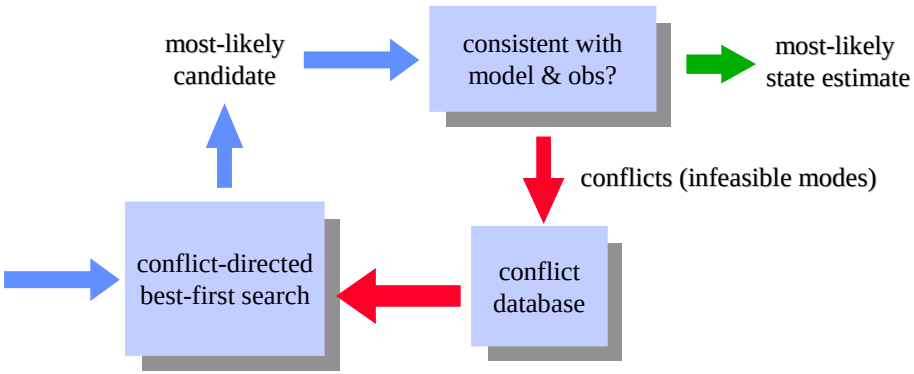


Overview

- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Computationally Tractable State Estimation & Fault Diagnosis

- Recast Belief State Update problem as an Optimal Constraint Satisfaction Problem (OCSP)
- Solve using OPSAT engine:



Approximations to Estimation

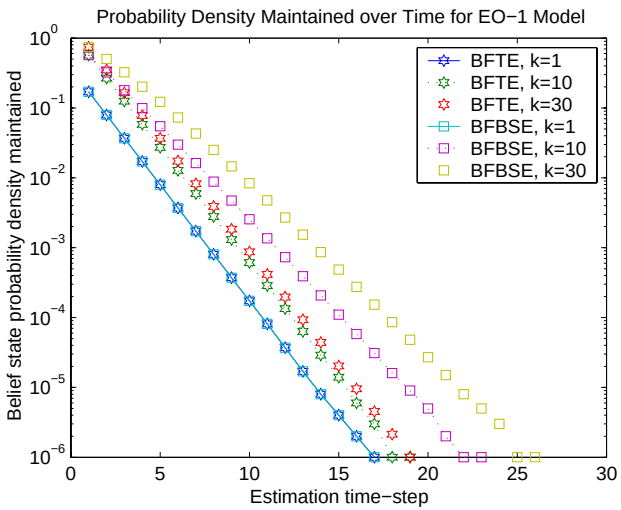
- The belief state can be accurately approximated by maintain the ***k most likely estimates***
- The probability of each state can be accurately approximated by the ***most likely trajectory*** to that state
- The ***observation probability*** can be reduced to
  - 1.0 for observations consistent with the state,
  - 0.0 for observations inconsistent with the state

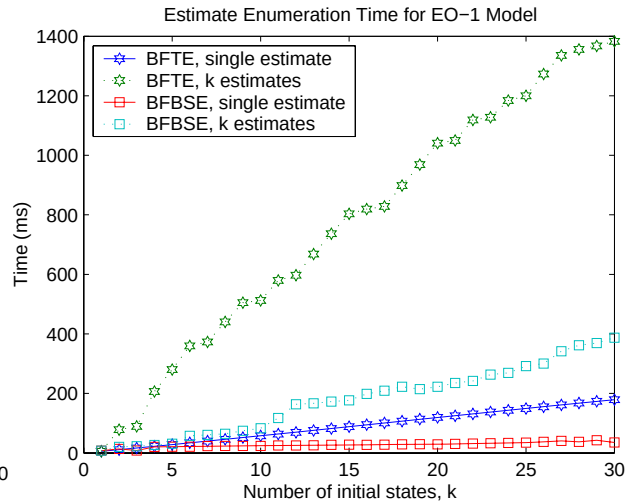
Assumptions made by Livingstone, Livingstone-2

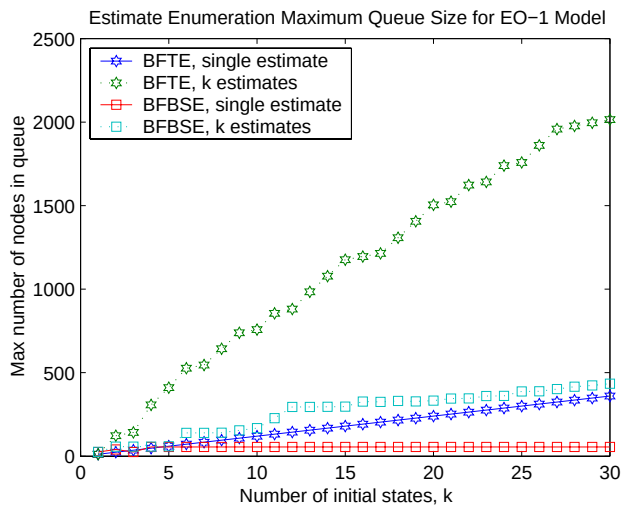
Assumptions relaxed by Titan Model-based Executive

Diagnostic Algorithm Performance

- Two primary concerns regarding application of model-based reasoning onboard spacecraft are *processor throughput* and *memory limitations*
- To address this concern, we have evaluated the performance of Titan's state estimation capability in terms of estimation time and static/heap memory utilization
  - Variety of models (up to subsystem-size)
  - Variety of nominal and off-nominal scenarios (auto-generated)







Integration and Test Issues

1. Reconciling Titan's *approximate belief state* representation with the MDS State Variable concept?
   1. Can collapse all State Variables into a single 'System State Variable'
   2. Retain individual states by computing each state's estimate by adding up probability of each system state estimate that contains the specified state estimate (preferred)
2. How to integrate Titan with parts of the system which deal with continuous states?
   1. Allow Titan to handle discrete states, and write traditional estimators for continuous states.
   2. Research into *hybrid model-based programming* environments (possible direction for future work)
3. Titan improperly diagnosing FAILED states
   1. Titan does not properly handle some types of dependencies between state variables. Future work will need to characterize exactly which type of dependencies are problematic and possibly modify the existing Titan algorithm, as well as develop a set of modeling guidelines to ensure that the models properly express the physical dependencies of the system under control

Demonstrated Benefits

End-to-end Lifecycle Approach	• Model-based systems engineering at the core • Earlier consideration of off-nominal behavior and operations • Regular architecture allows development of accurate cost and performance models
Simplified estimator & controller development	• Current ad-hoc algorithm implementation reduced to model development and use of pre-validated general estimation & control algorithms • Greater s/w reuse: only application-specific engineering models need to be developed; the reasoning engine is generic and reusable
Enhanced diagnosis and recovery capabilities	• Uses complete, any-time inference algorithm to detect and respond to failures “on-the-fly” • Increased runtime performance using a precompiled model • Reasons over component interactions and “hidden” states • Tracks multiple state possibilities; prunes inconsistent states
Inspectable & verifiable models	• Models are specified using an intuitive graphical representation (Factored POMDP) • Models are compiled offline into a set of rules that can be verified by systems engineers • Formal modeling semantics provides greater opportunity for automated verification (e.g., model-checking)

Overview

- Introduction
  - Motivation
  - Objectives
- Background
  - MDS
  - Model-based Programming and the Titan Executive
- Detailed Approach
  - Compatibility Analysis
  - Framework Development
  - State Analysis and Modeling
  - Adaptation Development
  - Testing
- Results
- Conclusions and Future Work

Next Steps

- Demonstrate integrated architecture and approach on scaled-up system
- Accommodate other modeling representations and estimation algorithms, e.g., hybrid systems (discrete modes with continuous dynamics)
- Explore applicability of model-based programming to other domains of interest (e.g. planning & scheduling)
- Develop formal V&V approach for model-based programs
  - probabilistic analysis of possible system executions
  - lead to verifiably-correct system behavior for both nominal and off-nominal execution
  - focus on specific validation needs of future NASA exploration missions