



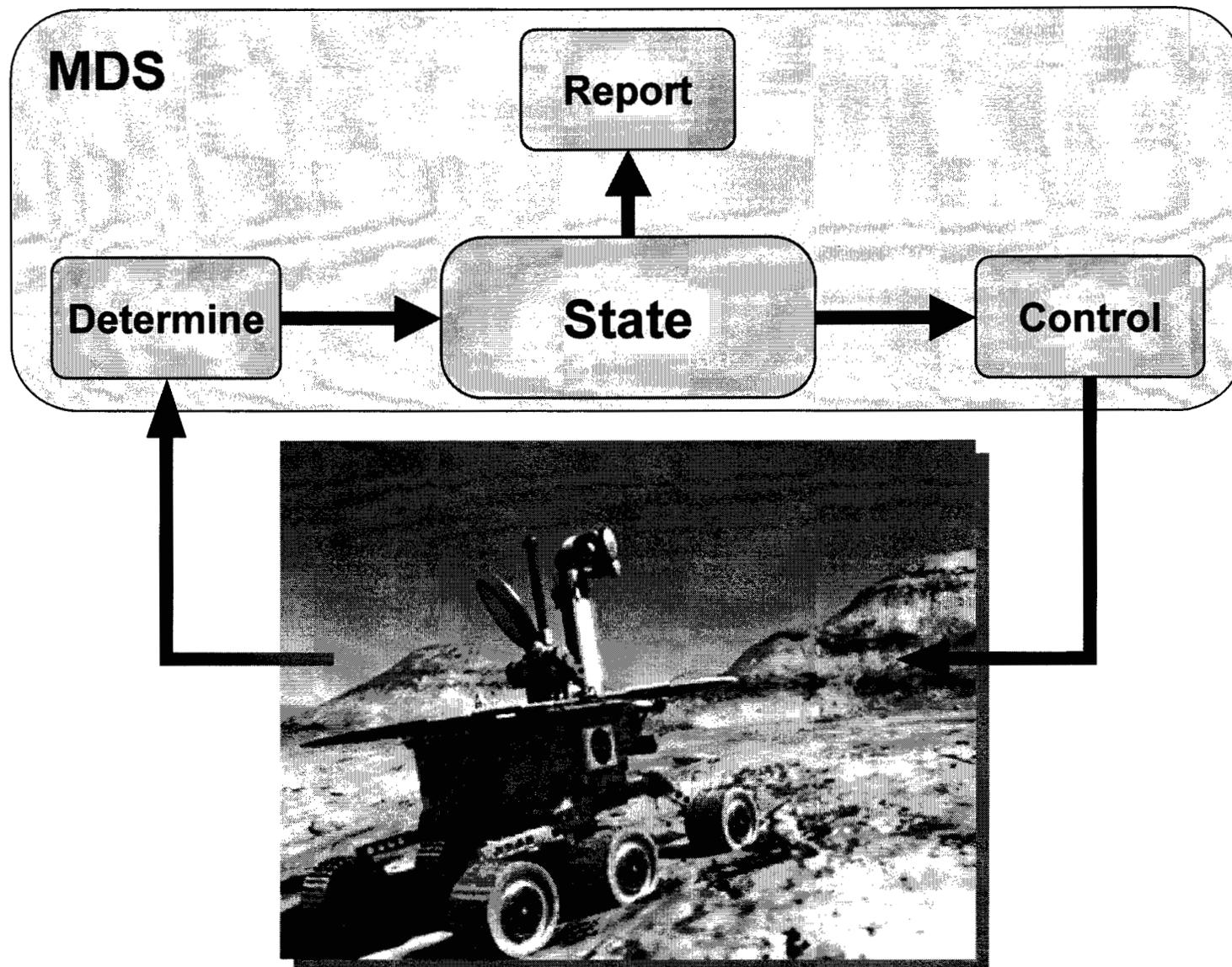
MDS

State-Based Architecture

Bob Rasmussen

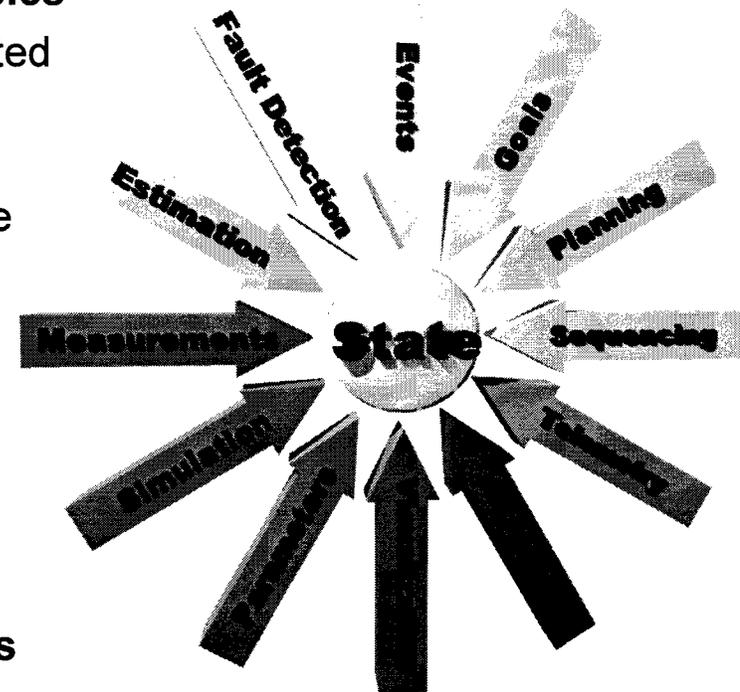
May 16, 2001

State-Based Architecture

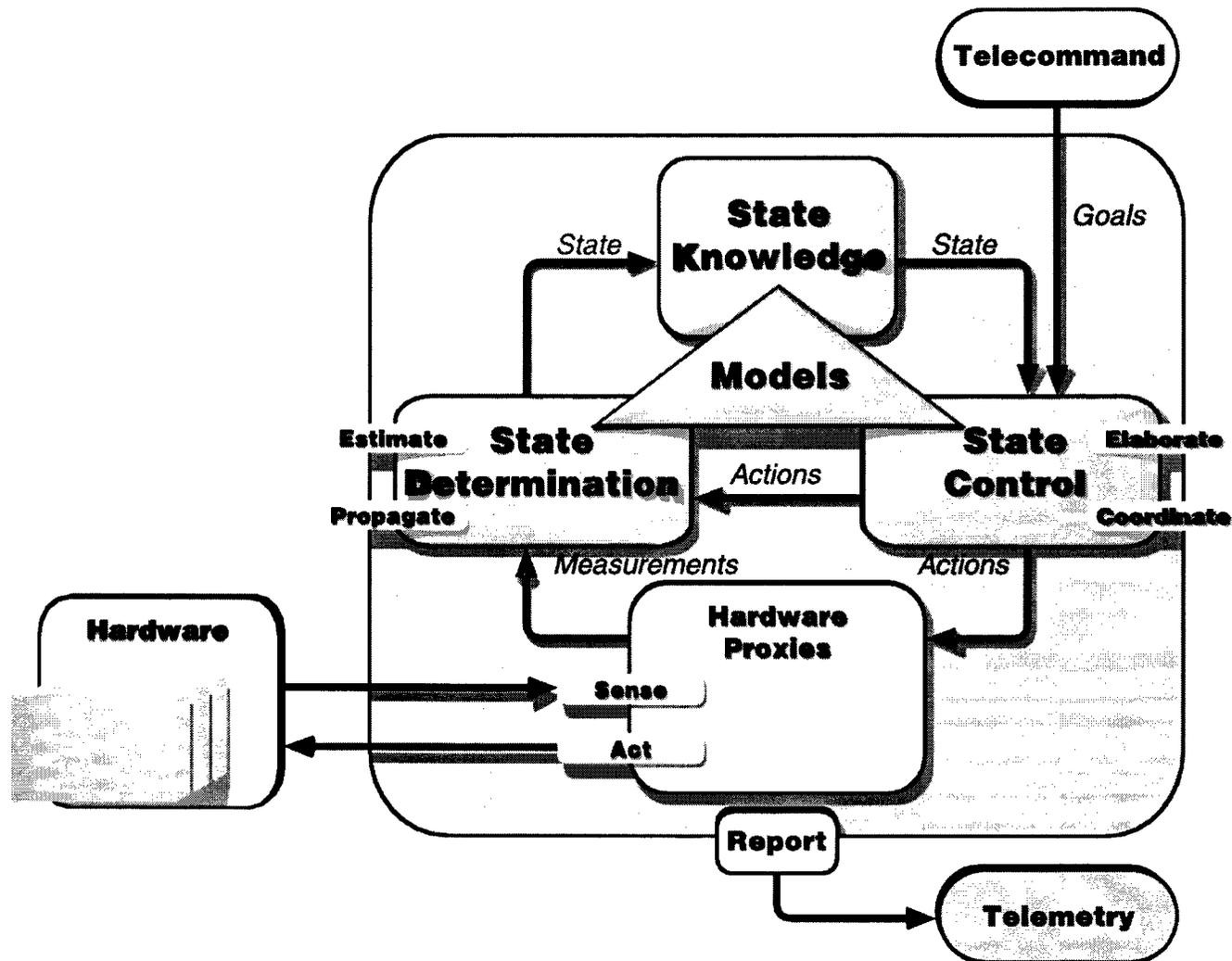


State is Central

- 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



A High Level View





Types of State

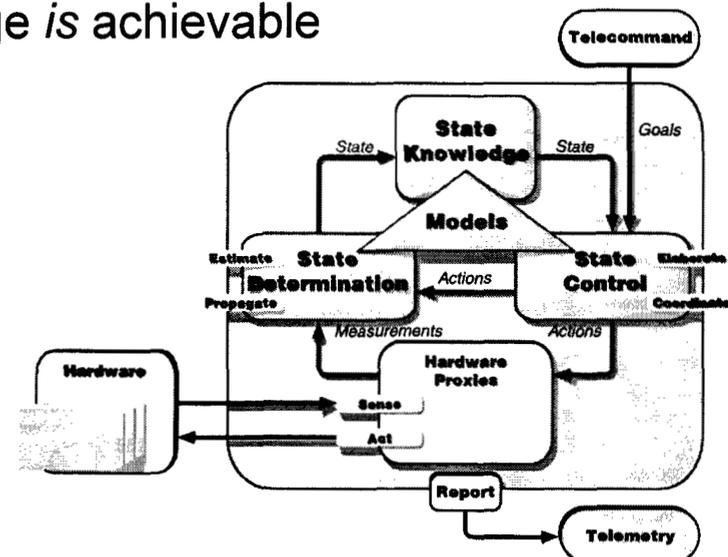
- Dynamics
 - Attitude, position, gimbal angles, altitude, ...
- Environment
 - Ephemeris, light level, atmospheric profiles, terrain, ...
- Device status
 - Configuration, temperature, operating modes, failure modes, ...
- Parameters
 - Mass properties, scale factors, biases, alignments, noise levels, ...
- 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

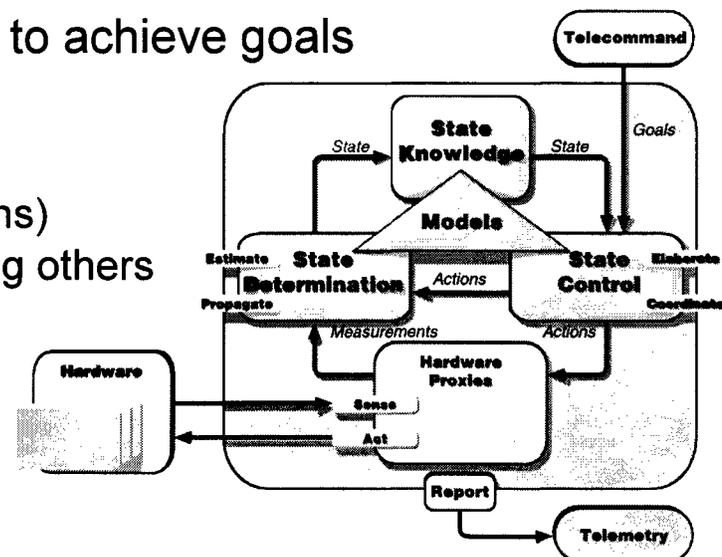




State Control Closing the Loop

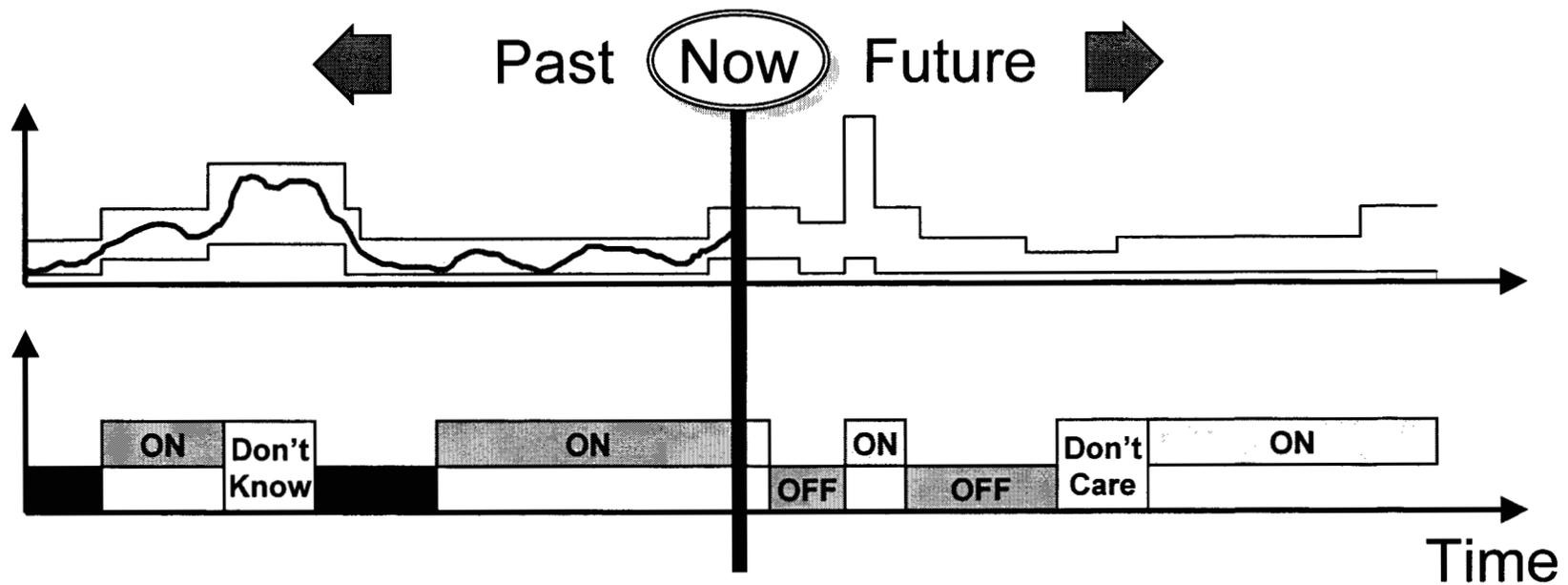


- Operators express their intent in the form of **goals**
 - Goals declare **what** should happen, **not how**
 - Goals may be expressed at any level
- High level goals are elaborated recursively into lower level goals
 - **Elaboration** may be conditional, in order to react to present circumstances
 - **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**

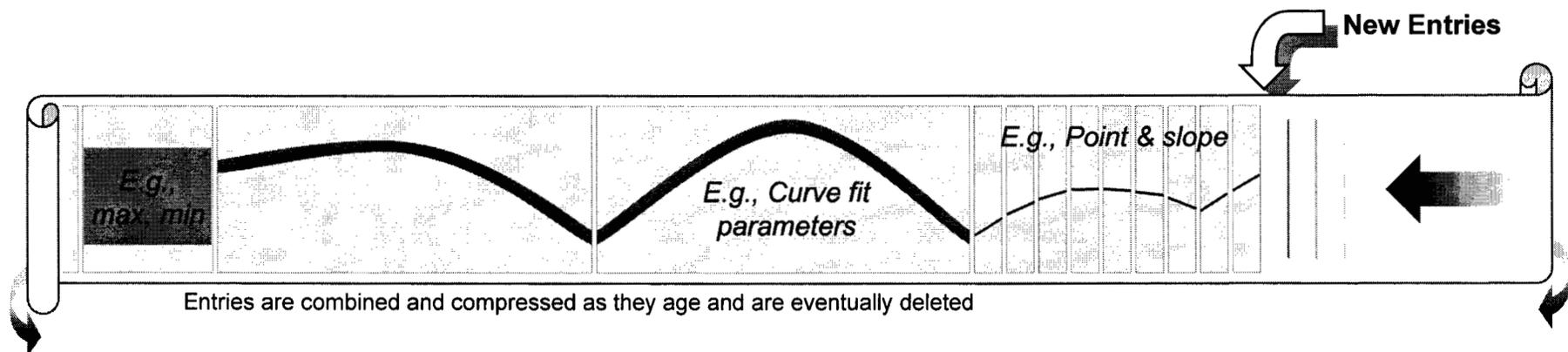


State Timelines

- **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



- 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**





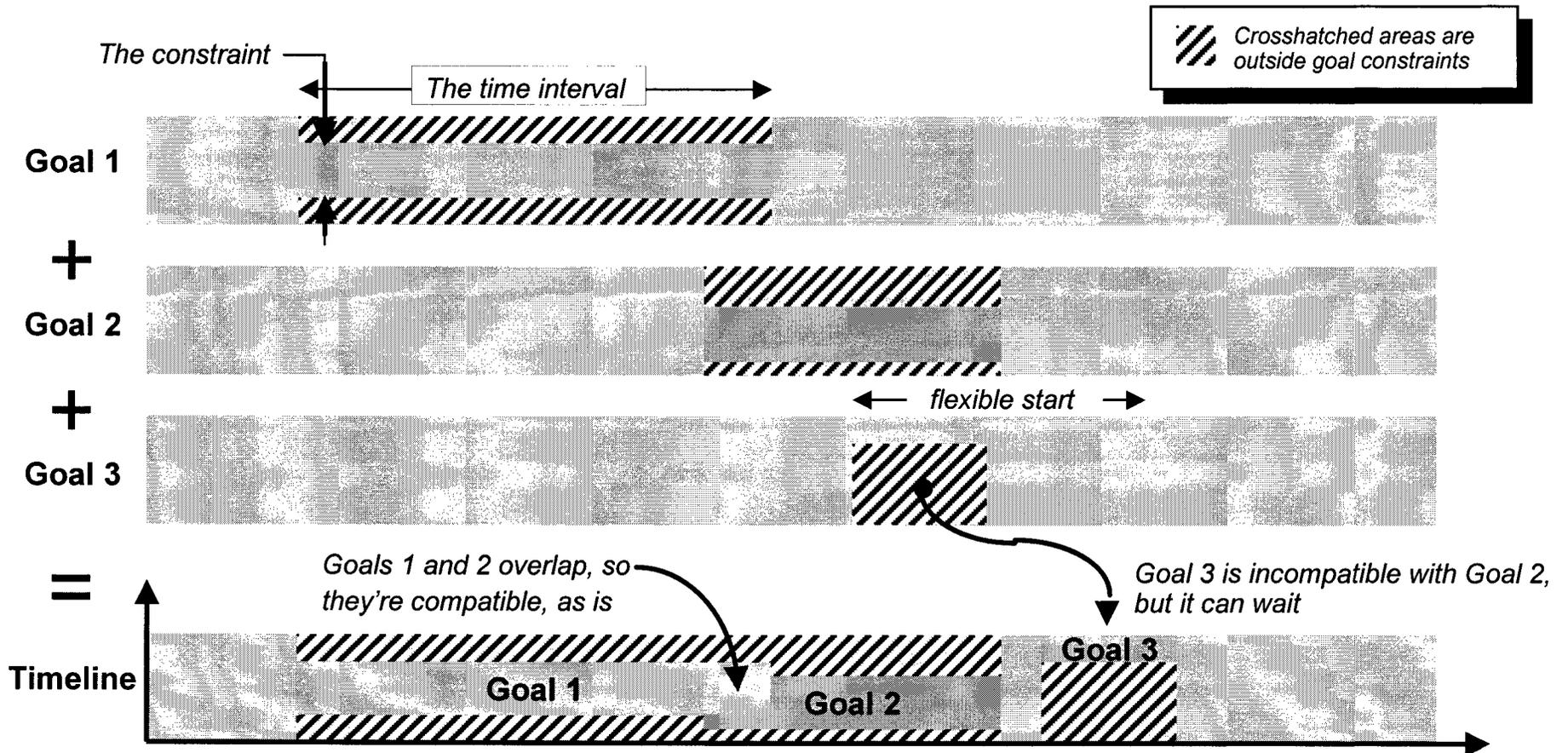
State and Time Constraints



- Control is exercised 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

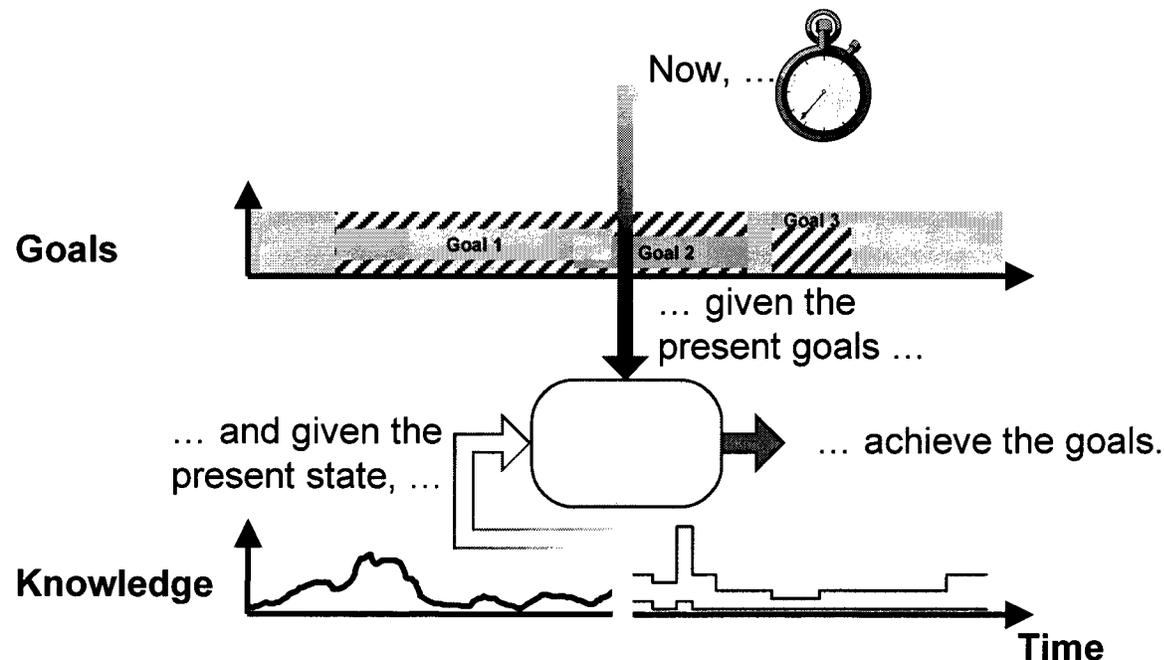
Resolving Conflicts

Three goals on the same state



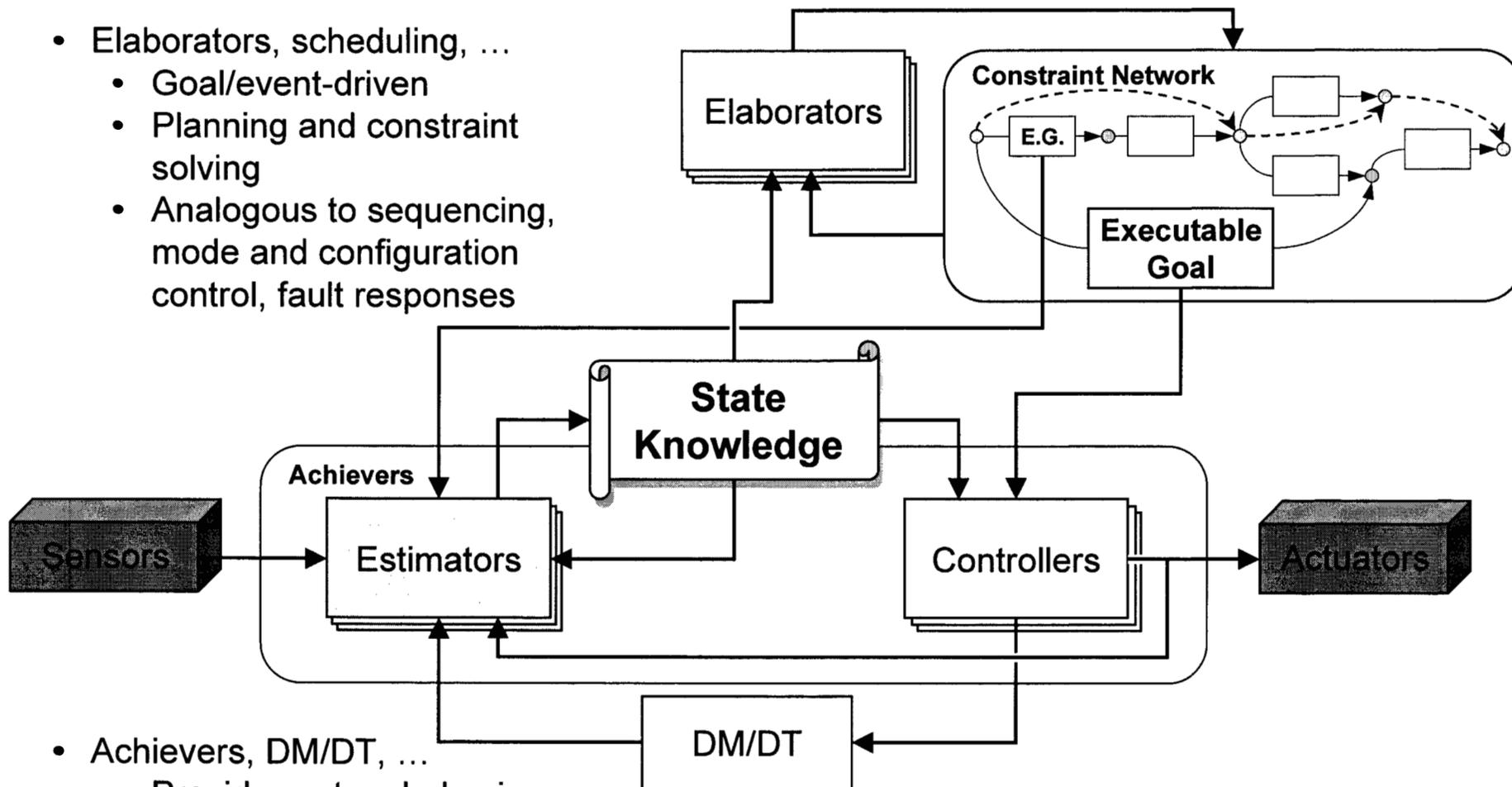
Timeline Execution

- 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



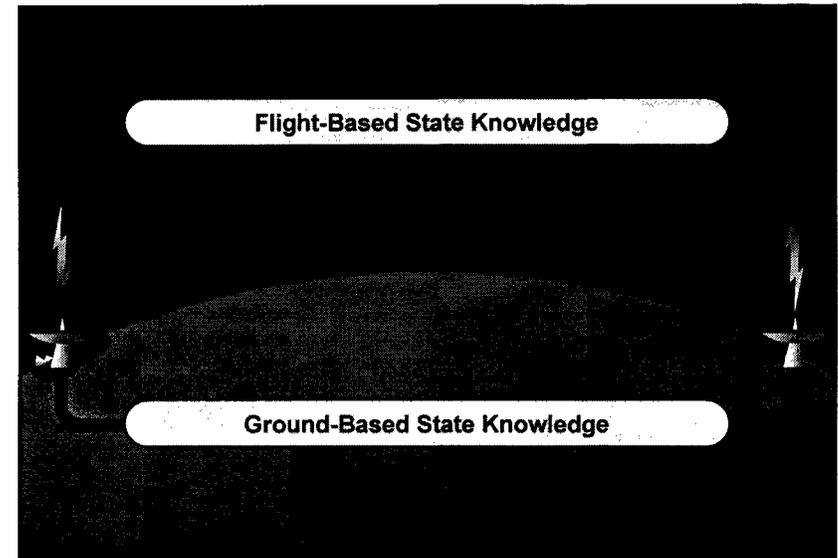
Putting It Together

- 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

- 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:
- http://mds-lib.jpl.nasa.gov/mds-lib/dscgi/admin.py/Get/File-6391/EDL_Workflow_04_25_01.xls
- 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

	Increment	Simulation	Flight	Transport	Ground	Framework	Test
	EDL "Rock"	Start with simple vertical descent within aeroshell beginning about 10 minutes from impact					
1	Get initialized in a free fall	Point mass with no rotation dynamics (assume fixed upright attitude)	Perfect time knowledge One state variable (scalar continuous value with	Start with just a wire link Transport uncompressed altitude speed state history	Perfect time knowledge Proxy for vehicle altitude state variable initialized to "unknown"	Dynamics simulation Simulation generated value histories	Show that plots of "unknown" values show "unknown" (which is different from missing data)
2	Update altitude from ground estimates		Update vehicle altitude and descent rate estimate via ground provided data Report update event (presumably as an ELF message) Extend estimate of altitude and descent rate by integrating trivial dynamics model past end of ground provided estimate function	Transport altitude estimate from ground to flight Transport update event message, flight to ground	Prepare transportable estimate for vehicle altitude and descent rate over some interval that includes the present Display update event message	Manual preparation and submission of an estimate function (i.e., human-in-the-loop estimator) Policy recognition that an estimate function needs to be transported Value history transport via wire protocol ground to flight ELF event transport via wire protocol flight to ground	Compare simulation and ground state history results, which should match to within accuracy of manually prepared update for the portion of value history that was updated
3	Make position 3-dimensional	Switch from altitude-only to a 3-dimensional position, one coordinate of which is altitude Gravity is still strictly rectilinear and vertical	Change vehicle altitude state variable into 3-dimensional position state variable (2nd-order polynomial estimate function for each axis) Assume independent axes for now -- still zero uncertainty	Transport 3-dimensional position flight to ground	Change vehicle altitude proxy state variable into 3-dimensional position proxy state variable Extract altitude/speed from 3-dimensional position Co-plot altitude/speed from simulation and transported state history	State variables with vector interval value histories (mean and sigma for each)	Repeat uplink and downlink steps above with new 3-dimensional state variable
4	Add Mars surface	Flat, smooth, horizontal plane at some altitude with no hazards No descent below this point Stop flight system and simulation if velocity at impact is too large	Add surface altitude state variable (altitude with uncertainty above some ideal reference surface as vehicle altitude) Surface altitude recovered at initialization is a fixed value Constant estimate function (i.e., 0th order polynomial) Accompany ground update of vehicle altitude with surface altitude update Modify vehicle altitude estimator to project impact at surface (i.e., no descent below surface altitude)	Transport estimated surface altitude, ground to flight	Add proxy for surface altitude state variable Prepare transportable estimate update for surface altitude Display fixed value Mars surface altitude as text	Ability to stop flight system on detected simulation condition Textual value display of scalar values from state variable query (since this is a one-shot item, time really isn't a factor yet)	Verify Mars surface update Verify sim and flight system stop at impact while ground system keeps running (reality therapy)
5	Estimate altitude above Mars surface		Add a derived state variable for vehicle altitude above Mars surface	(Derived states are not transportable)	Add a derived state variable for vehicle altitude above Mars surface Plot derived altitude/rate	Derived state variables MONTE translation frame tree (with the ideal reference surface as root, vehicle altitude above ideal reference surface as one branch, and Mars surface altitude above ideal reference surface as a separate branch)	Show simulated impact occurs near where derived altitude above Mars surface says it should (i.e., at about zero)



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



Add measurements, estimators, fault detection from measurements



	Add ideal accelerometer	<p>device</p> <p>One linear vertical axis acceleration accumulating accelerometer (no bias or noise for now)</p> <p>Sample time controlled by flight software (assume fixed delta-t for now with delta-v approximately acceleration times delta-t)</p> <p>Should show zero g</p>	<p>history</p> <p>Add integration of acceleration measurements to estimator (effecting only vertical axis)</p> <p>NOTE—Properly done, the measurement supplies residuals via a measurement model, which the estimator integrates along with model'd acceleration. This is important for later.</p> <p>Create a residual value history for text purposes (being able to tap internal calculations like this may be a common request)</p> <p>Default transport policy to send all</p> <p>Default storage policy to keep only the last value</p>	<p>acceleration measurement history, flight to ground</p> <p>Transport uncompressed residual history, flight to ground</p>	<p>measurements (as delta-v for now)</p> <p>Plot measurement residuals (not by asking measurements, since the a priori estimate is no longer available, but rather by getting them directly from the estimator)</p> <p>Default storage policy to keep all</p> <p>Default transport policy to send none</p>	<p>(basically a direct exercise of the measurement model, but with actual/simulated state as input)</p> <p>NOTE—Maintain ability to simulate at this level as an option (independently selectable on each interface), regardless of subsequent detailed simulation capability</p> <p>Measurement model available from measurements</p> <p>Accurate measurement time available from measurements</p> <p>Plot interface to general value histories (measurements, residuals, and simulated values in this case)</p> <p>Plots should show values only at discrete times (no interpolated fill)</p>	<p>measurements (value and frequency) arrive</p> <p>Show measurement values and residuals are both zero</p> <p>Show no change in estimates</p>
7	Add atmospheric drag	<p>Model as vertical force versus drop speed (no winds, turbulence, no change in density with altitude, ...)</p> <p>Accelerometer should sense non-gravitational acceleration (deceleration in this case)</p>	<p>Add a drag model to the vehicle position estimator (no process noise)</p> <p>Assume built-in, fixed drag force and vehicle mass parameters that match the simulation</p> <p>Add non-gravitational acceleration to the accelerometer measurement model (the vehicle position estimator must make this available as an additional "view")</p>		<p>Plot non-gravitational acceleration</p> <p>Plot measurement residuals</p>	<p>Model sharing (estimator shares its model of non-gravitational acceleration via an additional view on the vehicle position state)</p>	<p>Show estimate and simulation still track</p> <p>Show accelerometer measurements not zero and match simulation, but residuals are still zero</p>
8	Fail accelerometers	<p>Stop measurement output at a specified time part way through the scenario (as dictated in the text script)</p>	<p>Add accelerometer "status" state variable (just OK or FAILED as value range for now)</p> <p>Init status to UNKNOWN (= OK or FAILED)</p> <p>Estimate accelerometer status as OKAY if there are measurements, UNKNOWN after one or more misses, and FAILED (trap state) if more than three misses</p> <p>Update acceleration measurement model to include status</p> <p>Vehicle position estimator should screen bad measurements and continue to run (Alternatively, it should subscribe to "sanitized" measurements from the accelerometer status estimator and continue to run with no measurements)</p> <p>* "Sanitized" means that the values of some states in the original measurement model have been committed in order to present a new measurement with a simpler model to downstream estimators.</p>	<p>Transport uncompressed accelerometer "status", flight to ground</p>	<p>Add proxy for accelerometer "status" state variable</p> <p>Display current accelerometer "status" as text (this means there is a running display time clock driving the display, so there must be some way to control delay and sampling rate)</p> <p>Alarm to operators on accelerometer failure (this means there must be some way to control alarm criteria)</p>	<p>Sensor error reporting to estimator, where missing data in this case is reported as a measurement with no data</p> <p>Ability to order processing (in this case accelerometer status estimate runs before position estimator so data from bad accelerometer data is "screened")</p> <p>Discretely valued, intervalic value history</p> <p>Value sets (for enumerated types)</p> <p>Real time textual value display of value set values from state variable queries</p> <p>Simulation supports fault injection</p> <p>Alarms in ground system</p> <p>* Send measurement to both vehicle position and accelerometer status estimators (Alternatively, only the accelerometer status estimator sees the raw measurements, and it produces new "sanitized" measurements for the position estimator whenever the accelerometer is declared OK)</p>	<p>Show estimate and simulation still track, despite failure</p> <p>Show accelerometer measurements report errors</p> <p>Show failure is reported correctly in accelerometer status</p>



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**

Controller & Goal Increments

16	Add a simple pyro switch	Commandable one-shot pulse, but no arming, and so on, yet No measurements of switch state Add test port interface and means to manually initiate switch command via the port	Add state variable for pyro switch state (value range of OPEN or CLOSED -- another value set) Add test port Add switch state controller (for test port to talk to) Initialize state variable to OPEN (though UNKNOWN - OPEN or CLOSED is defined) Estimate switch state from switch command	Transport compressed pyro switch state, flight to ground	Add proxy for pyro switch state variable Display current pyro switch state as text Alert to operators on change of state (this means there must be some way to control alert criteria)	Alerts in ground system (not the same as alarms, but rather a less alarming way to notify operators of something significant) Text interface for direct low-level commanding Command caves dropping as input to estimator (even if they come from the test interface)	Command switch (via test port on flight system) at some point above the surface Verify alert on state change at the correct time
17	Add supersonic chute	Triggered by a single pyro switch actuation (the one added above) Only new dynamics* is change in vertical deceleration versus drop speed at chute deployment (no need yet to carry apparent air mass, significant impulse associated with deployment, or other complications)	Add state variable for supersonic chute state (value range of STOWED, or DEPLOYED -- another value set) Initialize state variable to STOWED, (though other value sets are also defined) Deploy supersonic chute at fixed time (via a goal) Estimate supersonic chute deployment state from pyro switch state and acceleration state (from acceleration view on vehicle position, <i>not</i> accelerometer measurements)* Add supersonic chute state as an input to the altitude estimator (which means the estimator has to have a model of the chute drag) Ignore measurement with large residuals (note that timing mismatches may cause a short spike in the residuals at chute deployment; this should not be considered as incorrect unless persistent, but such measurements should be ignored for now) * Note that this is <i>not</i> a derived state	Transport compressed supersonic chute state history, flight to ground Transport goal net, ground to flight	Add proxy for supersonic chute state variable Prepare a transportable goal subnet and initiate transmission Default transport policy for goal subnets to send all marked as transportable Display supersonic chute state history Disable alert on switch state (this means there must be some way to remove alert criteria) Alert on chute deployment instead	Goal subnet with one goal and one zero-duration temporal constraint Goal subnet preparation tool Goal subnet transport (create transportable data product containing goal net) Goal subnet installation (into the initially empty flight goal net) on receipt Goal scheduling (fixed time for now, so this is essential null) Time point firing with directives to achieve (switch controller in this case) Monitor firing commands to determine precise command timing	Show goal net is received and installed Show switch commanded at the specified time (in the real system, this and many of the other timed events have to be done very accurately) Show supersonic chute state reported correctly Show accelerometer residuals remain small throughout, meaning estimator is anticipating chute effects
18	Fail the supersonic chute control switch	Set so commands to the switch have no effect	Add FAILED_OPEN to the switch state value range Accelerometer and switch status estimators should be able to tell the difference between a switch failure and a flat-line accelerometer failure Switch goal should report failed and terminate	Transport goal failure event, flight to ground (eventually, not all goal failures will be interesting and we'll need a way to say which ones are)	Update switch state variable proxy Update switch "status" display Alarm FAILED_OPEN state Display switch goal failure event Alarm switch goal failure event	Cyclically collaborating estimators (up to now estimators simply chained cyclically, as in switch --> chute --> altitude, but now estimated chute state can also affect whether or not the switch is estimated to be working) Goal failure handling by goal network Goal failure reporting Event alarms	Try both accelerometer and switch failure modes and show that the correct diagnosis is made Show correct alarms are reported
19	Make chute deployment switch redundant	Either switch will release the supersonic chute	Add second state variable for second pyro switch Put both switches in the same switch set "Item" Add second switch command to chute state estimator evidence Add second switch to subgoal net	Transport additional states, flight to ground	Add proxy for second pyro switch Display and alarm both switch states	Nested Items As a second instance of an existing class, transport, estimation, control, and so on should happen with little additional effort	Show chute can be deployed at the right time with one switch failed Show chute state is reported correctly



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

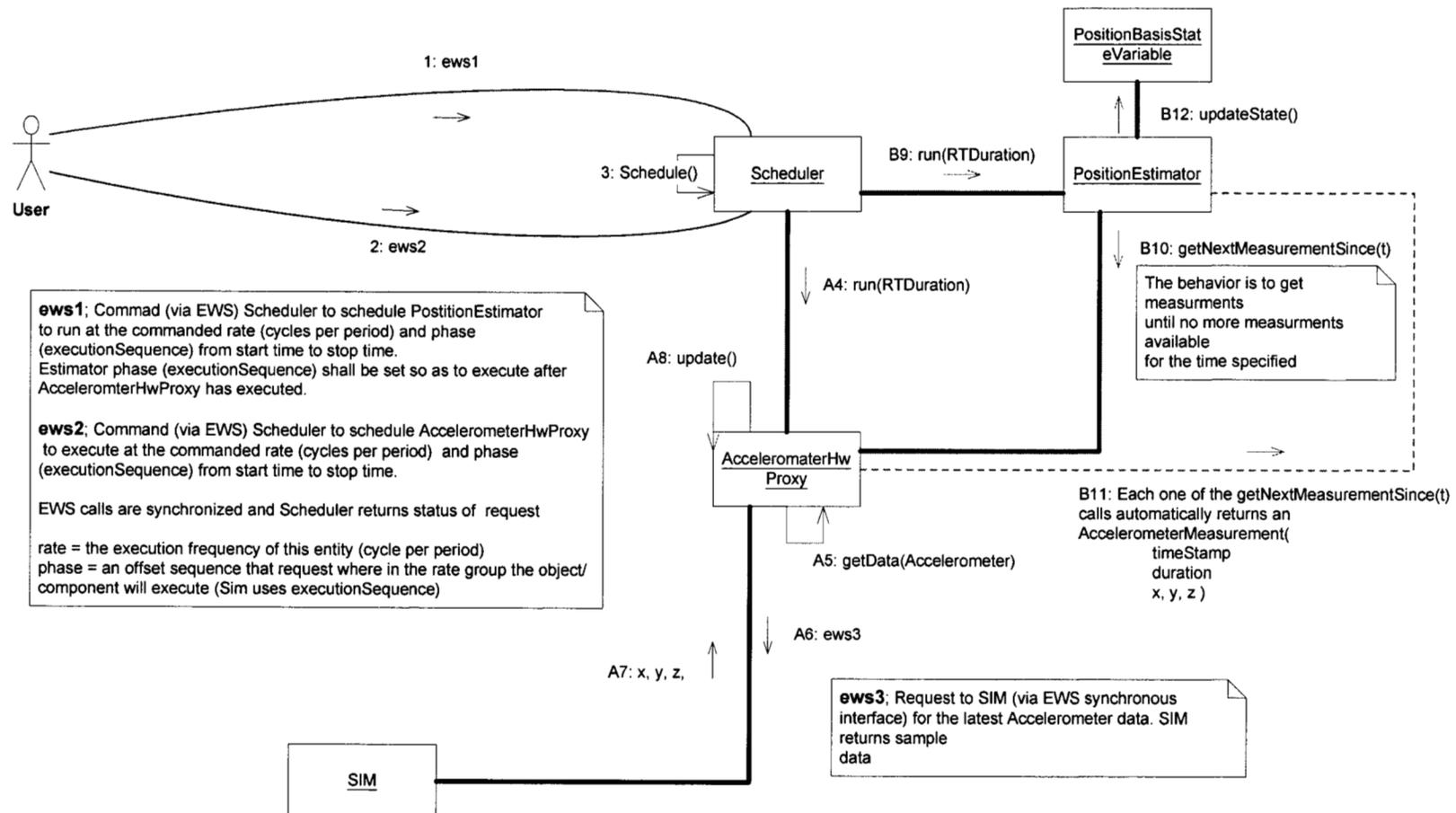


Component and Infrastructure Development

- Single-threaded components
- Multiple time frames
- Multi-threaded execution, synchronous and asynchronous communication
- Cyclic, time-alarm and event-based execution

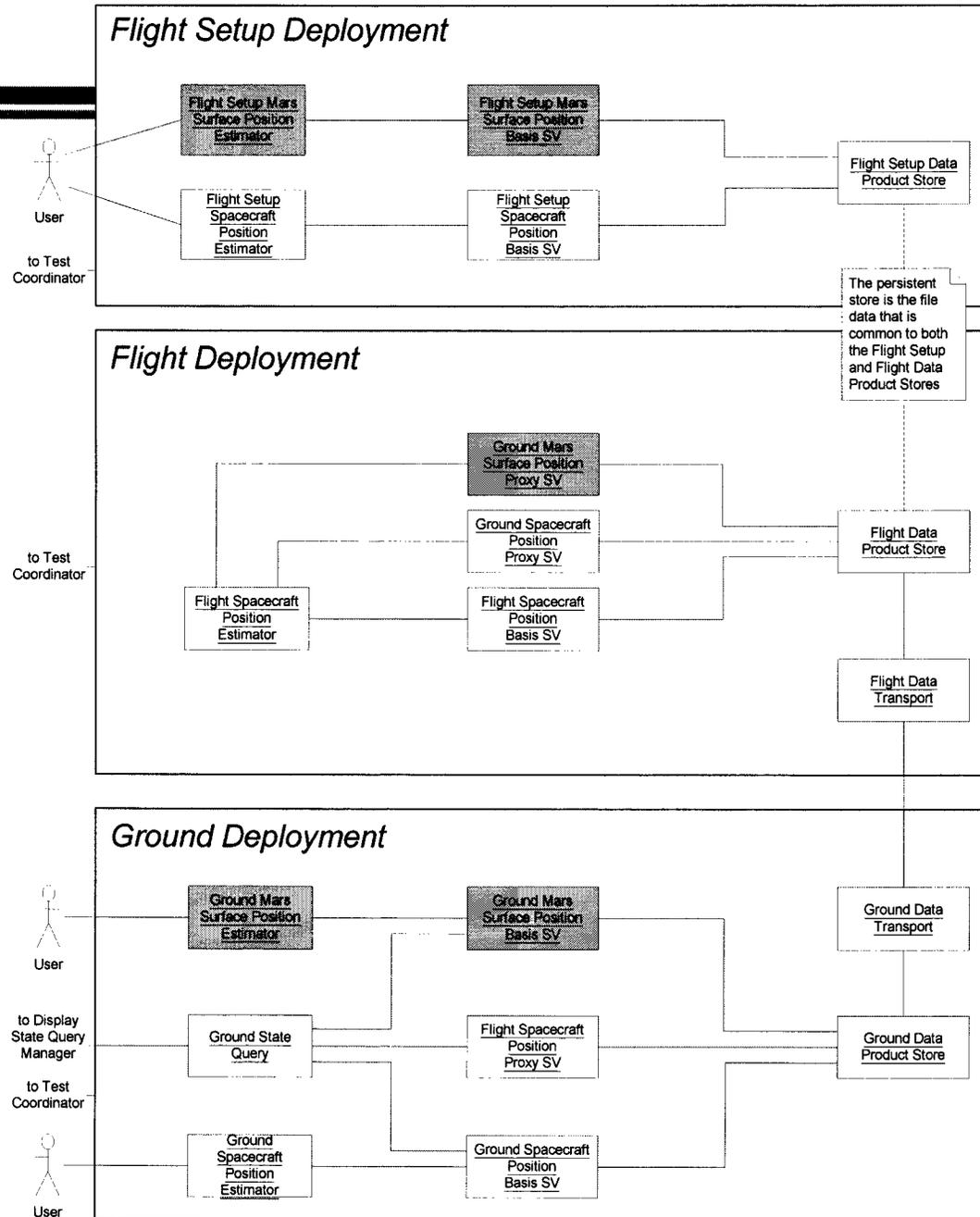


Identify Objects in the Increment





EDL Increment 4 Object Diagram

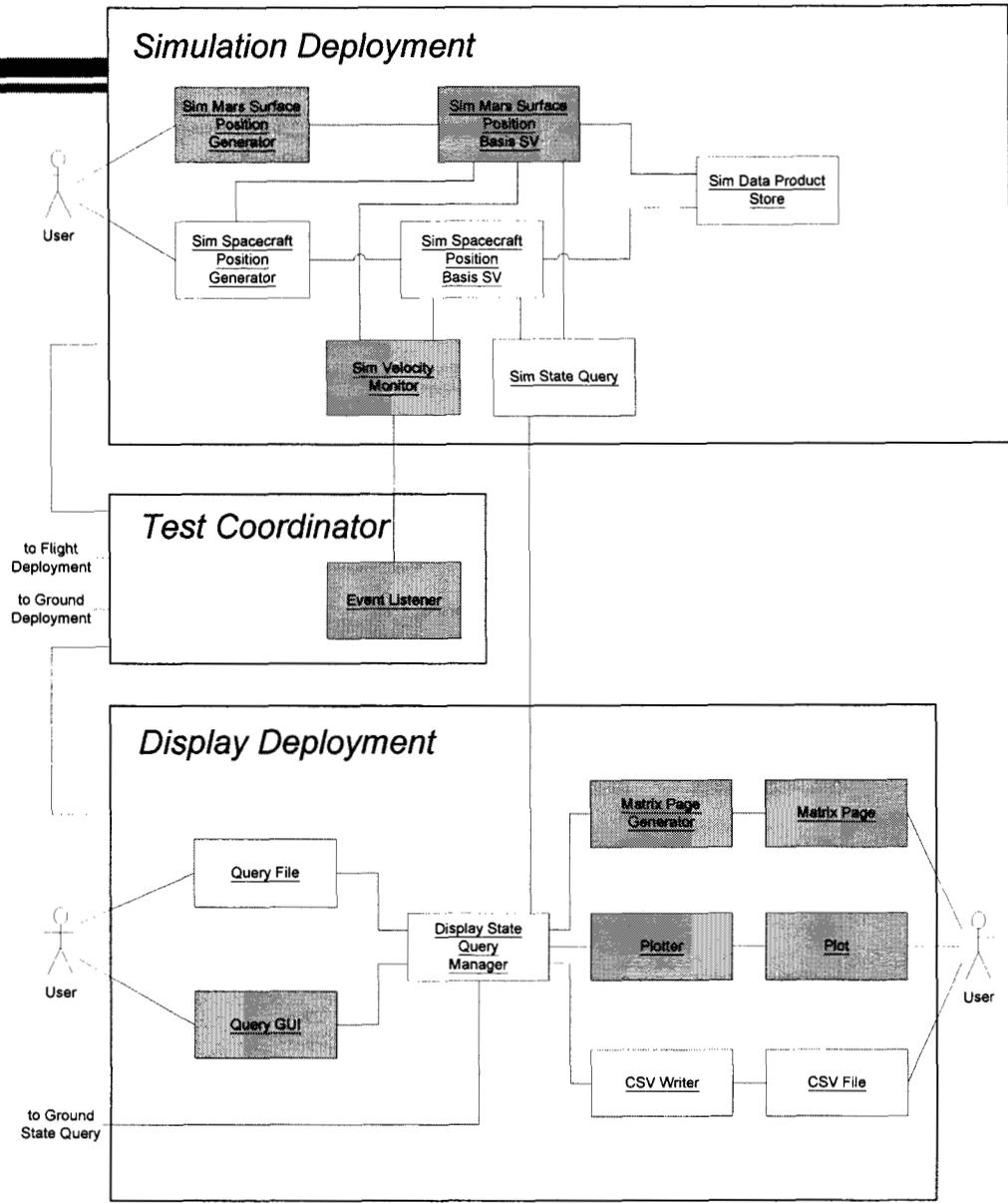


05/11/01

3/29/01



EDL Increment 4 Object Diagram



mds accelerometer increment-6 sequence diagram (Ver-H)

VClock AccelerometerHw Proxy Scheduler Position Estimator PositionBasis StateVariable SIM

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.
 Estimator 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

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 = ΔV_x (m/sec)
 Y = ΔV_y (m/sec)
 Z = ΔV_z (m/sec)

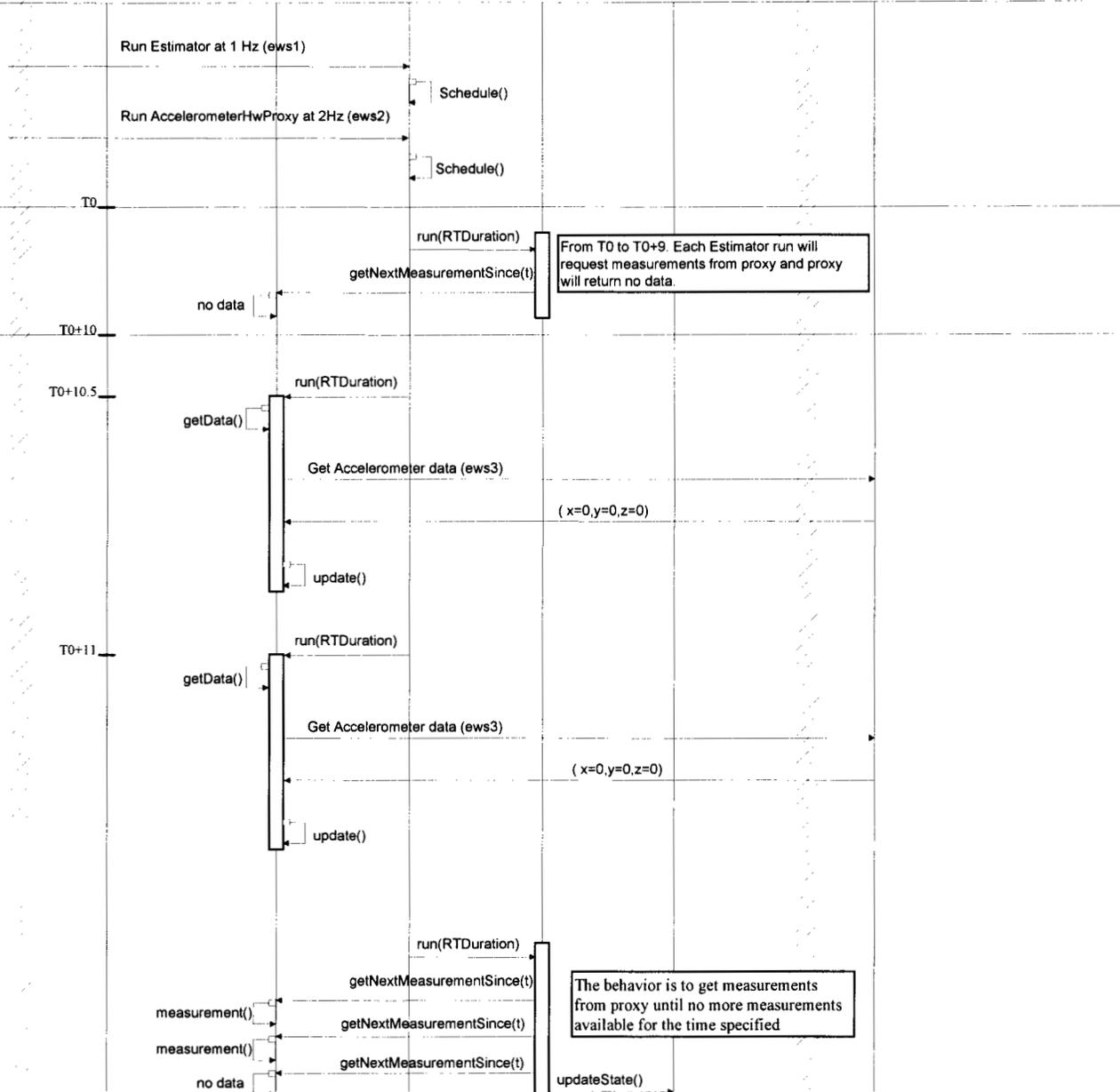
The device proxy automatically updates the measurement

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

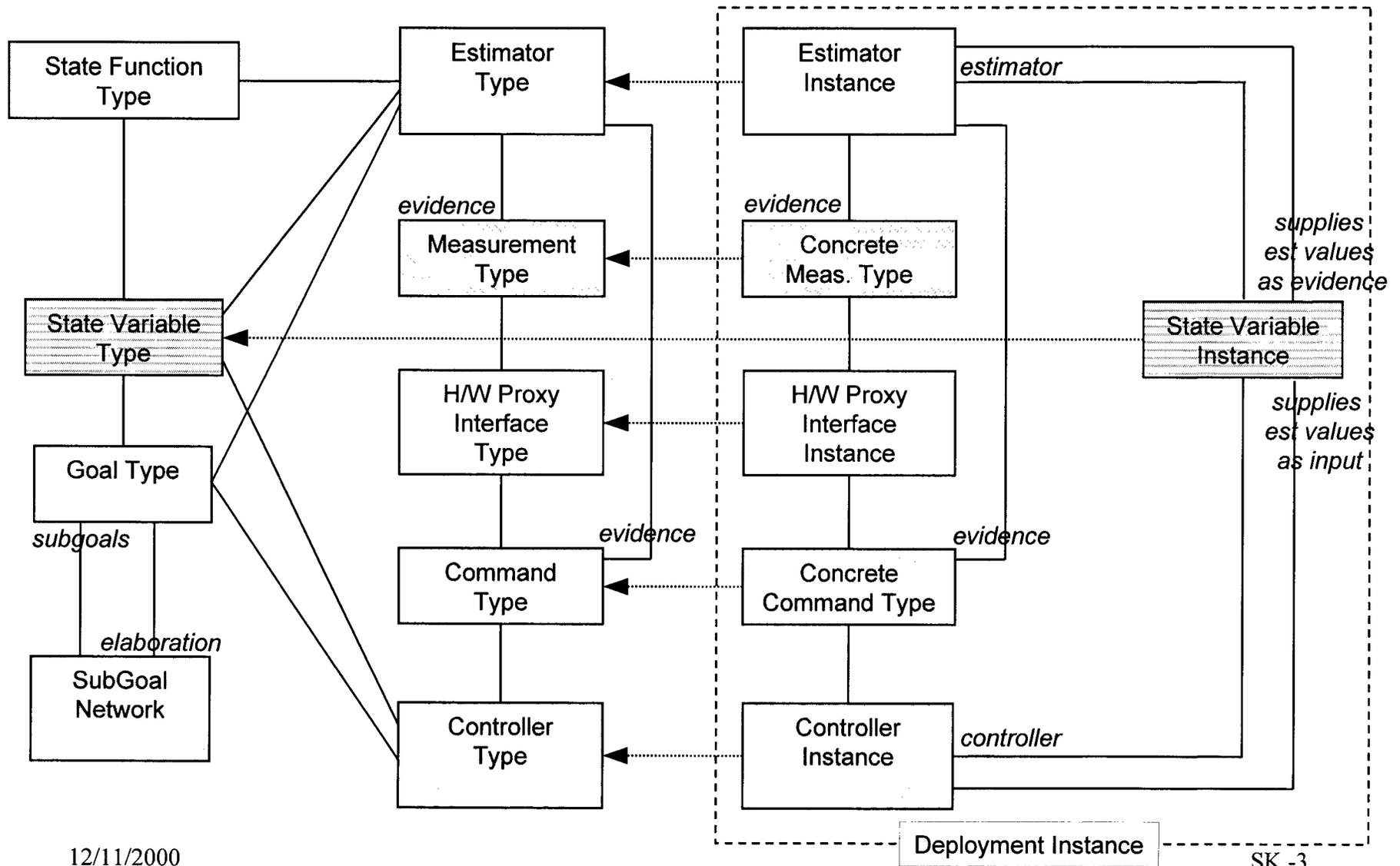


What is State Analysis?

- 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



State Database Relations



12/11/2000

May 10, 2001

SK-3



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



Bookmarks Location: http://sds/sds_beta_3/html_pages/sds_top.html

What's Related

Instant Message Members WebMail Connections BizJournal SmartUpdate Mkplace

[Login](#)
[New User](#)
[Beta Model](#)
[Alpha Model](#)
[Design Diagram](#)
[Req Doc](#)
[Release Notes](#)
[Help](#)

Mission Scenarios

- [SC-11747](#)
- [Increment 3 State Analysis - Take 1](#)
- [Rocky7 smart executive test 01](#)
- [29mar01 EDL](#)
- [MDS/EDL_03](#)
- [Rocky8 test sfp01](#)
- [MDS/EDL'03 \(Early March familiarization\)](#)
- [EDL](#)
- [SC-debugging](#)

Name	Edit	MDS/EDL'03 (Early March familiarization)
Description	Edit	This state analysis supports the MDS/EDL Reference Mission & Reference Spacecraft Definition. It defines the current best understanding of what the Spacecraft will look like at the end of FY03.
Container		BETA-ROOT-0
Owner		robert.c.barry@jpl.nasa.gov
Categories	Add Edit	GNC
Groups	Add Edit	Rover Bat A Temperature GNC S/C Basebody GNC Entries
Actuator Types	Add Edit	Reaction Thruster Heater Series/Parallel Switch Combination
Actuator Instances	Add Edit	POSX Aligned Reaction Thruster2 (POSX-NEGY) POSX Aligned Reaction Thruster1 (POSX-POSY) EDL/Rover Battery A 2nd Heater EDL/Rover Battery A 1st Heater
Command Types	Add Edit	Thruster fire Solid State Power Switch (X2000) Control
Controller Types	Add Edit	Temperature Controller
Controller Instances	Add Edit	EDL/Rover Battery A Temperature Controller EDL/Rover Battery B Temperature Controller
Estimator Types	Add Edit	Device Power Available Temperature
Estimator Instances	Add Edit	EDL/Rover Battery A Heater 1 Power Available EDL/Rover Battery A Bulk Temperature
Goal Types	Add Edit	Configure Power to a Device, Select Final Control Switch Maintain EDL/Rover Battery A Temperature

12/11/2000

SK -5



Bookmarks Location: http://sds/sds_beta_3/html_pages/sds_top.html What's Related

Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace

[Login](#)
[New User](#)
[Beta Model](#)
[Alpha Model](#)
[Design Diagram](#)
[Req Doc](#)
[Release Notes](#)
[Help](#)

Controller Instances	Add	Edit	EDL/Rover Battery A Temperatue Controller
			EDL/Rover Battery B Temperatue Controller
Estimator Types	Add	Edit	Device Power Available
			Temperature
Estimator Instances	Add	Edit	EDL/Rover Battery A Heater 1 Power Available
			EDL/Rover Battery A Bulk Temperature
Goal Types	Add	Edit	Configure Power to a Device, Select Final Control Switch
			Maintain EDL/Rover Battery A Temperature
Measurement Types	Add	Edit	PRT Voltage Measurement
Sensor Types	Add	Edit	PRT Temperature Sensor
Sensor Instances	Add	Edit	EDL/Rover Battery A Temperature Sensor
State Function Types	Add	Edit	Voltage-Temperature Calibration Curve
State Variable Types	Add	Edit	Trial State Type
			Altitude (distance, metres)
			Temperature, Deg. K
			Position (distance, meters)
			Attitude (quaternion, non-dimensional)
State Variable Instances	Add	Edit	Trial State 2
			Trial State
			Mechanical: Basebody: Attitude
			Mechanical: Basebody: Location
			Mechanical: Basebody: Altitude wrt Landing Site
			Mechanical: Basebody: Horizontal Velocity wrt Surface
			EDL/Rover Battery A Temperature
State Value Types	Add	Edit	Single Precision Floating Point
Subgoal Networks	Add	Edit	EDL/Rover Battery A Heater 1 Power Configuration
View Types (Views)	Add	Edit	Velocity View

12/11/2000

SK -6

STATE VARIABLE TYPE:

State Type Name
State Type Description
Basis/Derived
Derivation (*Conditional: only if a derived state variable*)
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)

- (Link-S) State Variable Instance: State Variable Instances – [list of state instances of this type]
- (Link-RO) 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.)
- (Link-S) State Variable Type: Dependent State Variable Types -[list of derived state variable types that use this state variable type in their derivation]
- (Link-S) Goal Type: Applicable Goal Types – [list of types of goals that can be applied to this type of state variable]
- (Link-S) 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)
- (Link-S) Command Type: Command Types – [list of types of commands which affect this state variable type]
- (Link-S) Command Type: Command Types– [list of types of commands whose effects model use this state variable type]
- (Link-S) Measurement Type: Measurement Types – [list of types of measurements which measure this state variable type]
- (Link-S) Estimator Type: Evidence Receivers – [list of estimator types which use this state variable type for estimation]
- (Link-S) Estimator Type: Estimators – [list of estimator types that may compute estimates of this state variable type]
Conditional: only if a basis state variable
- (Link-S) Controller Type: Controllers – [list of controller types which use this state variable type for control]

- (Link) 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-S) State Variable Instance: Proxy Instances – [list of proxies to this state variable instance]

Conditional: only if a basis state variable. Decorated with deployment.

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

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

(Link-S) 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.

(Link) 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

Key

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.

This is the start time point.

Item Name:
State Name:
Goal Name[parameters]

This is the end time point.

Goals to be elaborated may be executable or non-executable

Everything below this dashed line is created by elaboration of the goal above the dashed line.

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

This is a special time point called the "epoch". It is some universally understood fixed reference point in time (e.g., 1/1/1958). Absolute temporal constraints are specified as relative temporal constraints with respect to the Epoch.

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.

Before
July 4, 2076

Elaborations can introduce new goals. These are called subgoals.

All or part of an elaboration can be conditional

This is also the start time point.

Item Name:
State Name:
Subgoal Name[parameters]

This is also the end time point.

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

[minimum, maximum]

Condition

Executable goals are shown as rectangles

Item Name:
State Name:
Subgoal Name[parameters]

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:
- Precedes = $[0, \infty]$
 - Delay D = $[D, D]$
 - At least L = $[L, \infty]$
 - At most M = $[0, M]$

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

Elaborations can introduce new time points.

Arrowheads show the direction of relative time flow. This is usually drawn left to right, but needn't be since the arrows are unambiguous.

