Subsystem Hazard Analysis (SSHA)

- Examine subsystems to determine how their
  - Normal performance
  - Operational degradation
  - Functional failure
  - Unintended function
  - Inadvertent function (proper function but at wrong time or in wrong order)

  could contribute to system hazards.
- Determine how to satisfy design constraints in subsystem design.
- Validate the subsystem design satisfies safety design constraints and does not introduce previously unidentified hazardous system behavior.

Software Hazard Analysis

- A form of subsystem hazard analysis.
- Validate that specified software blackbox behavior satisfies system safety design constraints.
- Check specified software behavior satisfies general software system safety design criteria.
- Must perform on ALL software, including COTS.

State Machine Hazard Analysis

- Like system hazard analysis, software (subsystem) hazard analysis requires a model of the component's behavior.
  - Using code is too hard and too late.
  - Software is too complex to do analysis entirely in one's head.

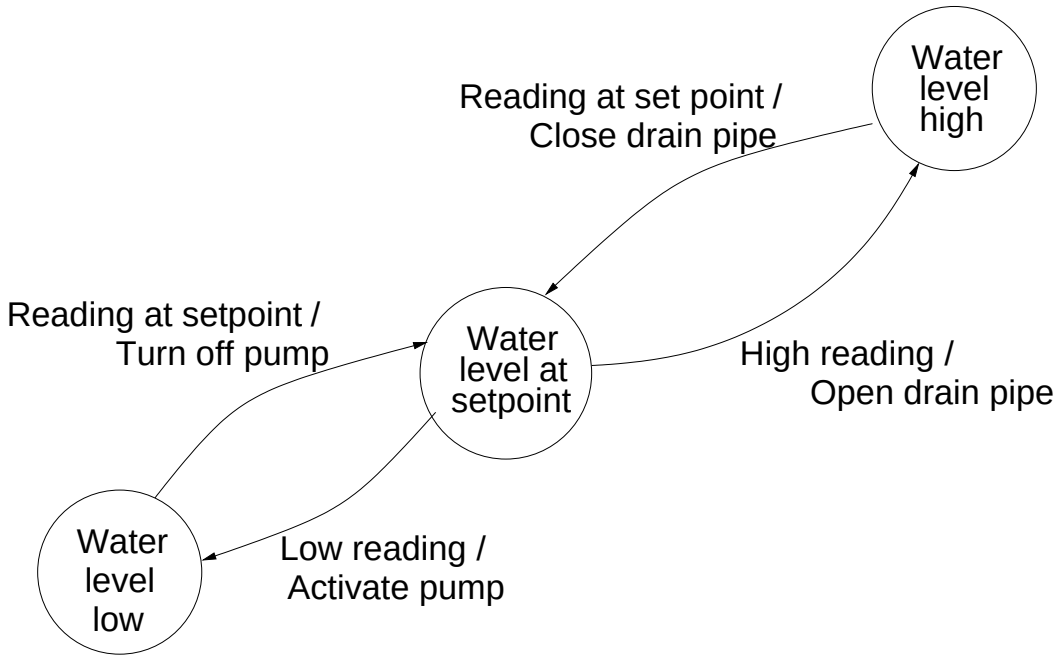
- Formal models are useful, but they need to be easily readable and usable without graduate−level training in discrete math.
  - Only a small subset of errors are detectable by automated tools: the most important ones require human knowledge and expertise.
  - Mathematical proofs must be understandable (checkable) by application experts.
  - Hazard analysis process requires that results can be openly reviewed and discussed.

State Machine Hazard Analysis (2)

- State machines make a good model for describing and analyzing digital systems and software.
  - Match intuitive notions of how machines work (e.g., sets do not)
  - Have a mathematical basis so can be analyzed and graphical notations that are easily understandable.
  - Previous problems with state explosion have been solved by "meta−modeling" languages so complex systems can be handled.

- Some analyses can be automated and tools can assist human analyst to traverse (search) model.
  - Our experience is that assisted search and understanding tools are the most helpful in hazard analysis.
  - Completely automated tools have an important but more limited role to play.

Example of a State Machine Model



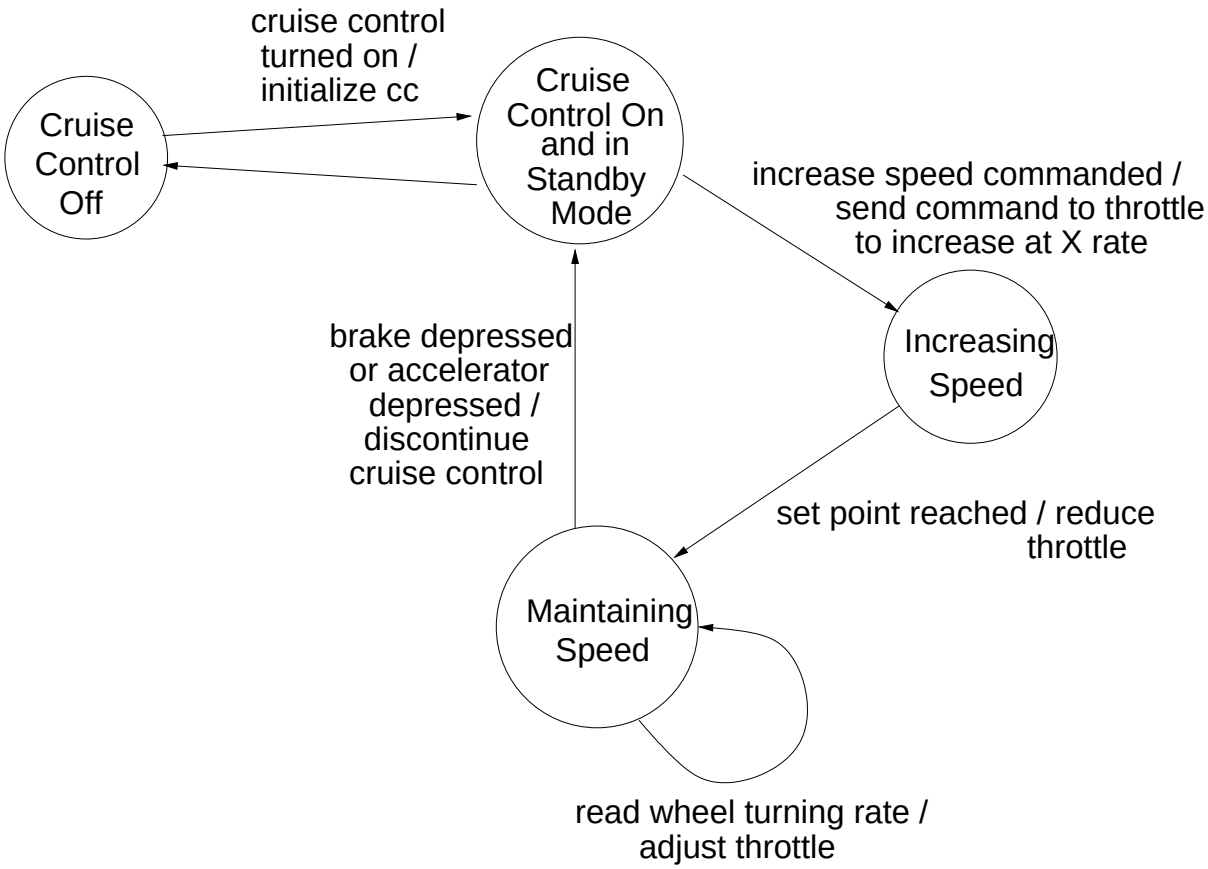
Requirements Validation

- Requirements are source of most operational errors and almost all the software contributions to accidents.
- Much of software hazard analysis effort therefore should focus on requirements.
- Problem is dealing with complexity

    1) Use blackbox models to separate external behavior from complexity of internal design to accomplish the behavior.

    2) Use abstraction and metamodels to handle large number of discrete states required to describe software behavior.

        - Do not have continuous math to assist us
        - But new types of state machine modeling languages drastically reduce number of states and transitions modeler needs to describe.


Cruise Control Off
cruise control turned on / initialize cc
Cruise Control On and in Standby Mode
increase speed commanded / send command to throttle to increase at X rate
Increasing Speed
brake depressed or accelerator depressed / discontinue cruise control
set point reached / reduce throttle
Maintaining Speed
read wheel turning rate / adjust throttle

Blackbox specifications

Provide a blackbox statement of software behavior:

Permits statements only in terms of outputs and externally observable conditions or events that stimulate or trigger those outputs.

trigger $\Longleftrightarrow$ output (double implication)

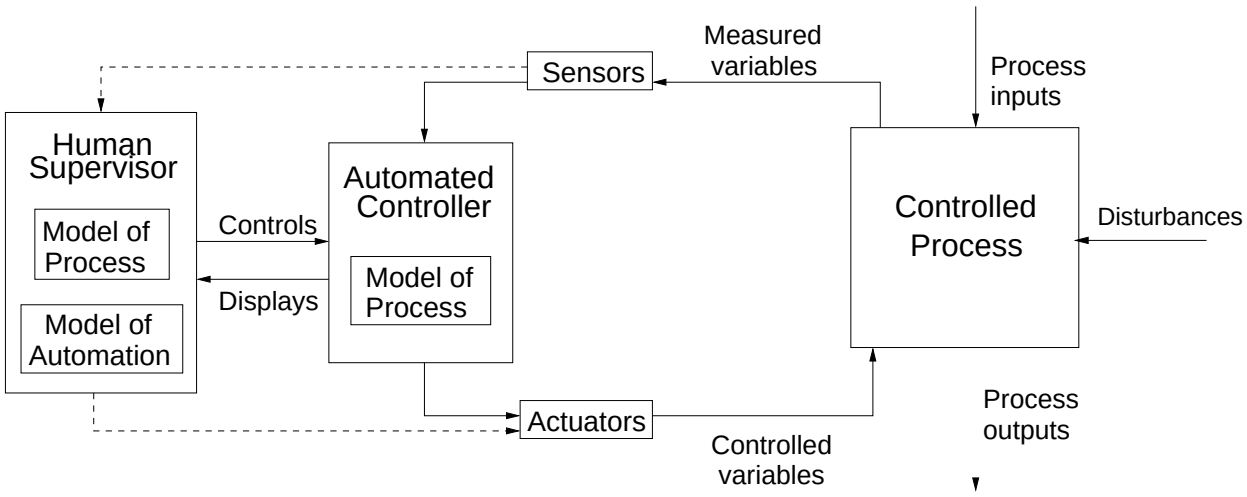
- Complete trigger specification must include full set of conditions that may be inferred from existence of specified output.
- Such conditions represent the assumptions about the environment in which program or system is to operate.

Thus the specification is the input to output function computed by the component, i.e., the transfer function.

Internal design decisions are not included.

Process Models

Define required blackbox behavior of software in terms of a state machine model of the process (plant).



Level 3 Specification (modeling) language goals

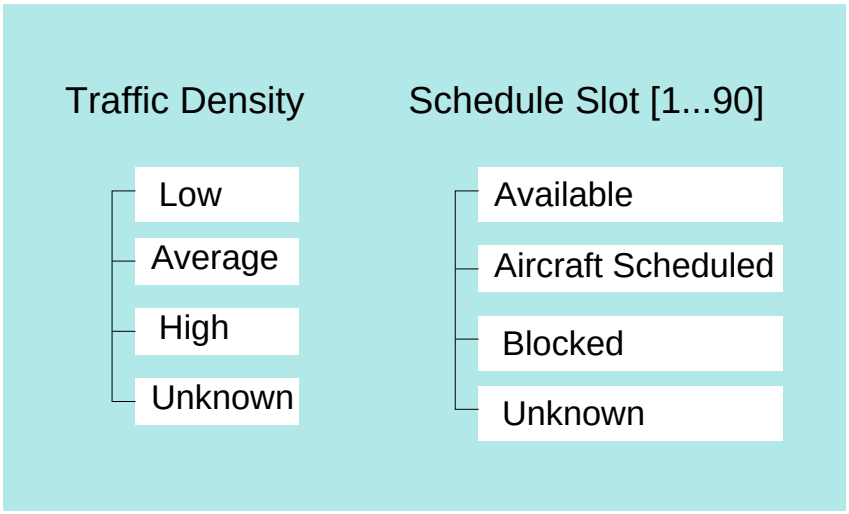
- Readable and reviewable
  - Minimize semantic distance
  - Minimal (blackbox)
  - Easy to learn
  - Unambiguous and simple semantics
- Complete
  - Can specify everything need to specify
- Analyzable
  - Executable
  - Formal (mathematical) foundation
  - Includes human actions
  - Assists in finding incompleteness

SpecTRM−RL

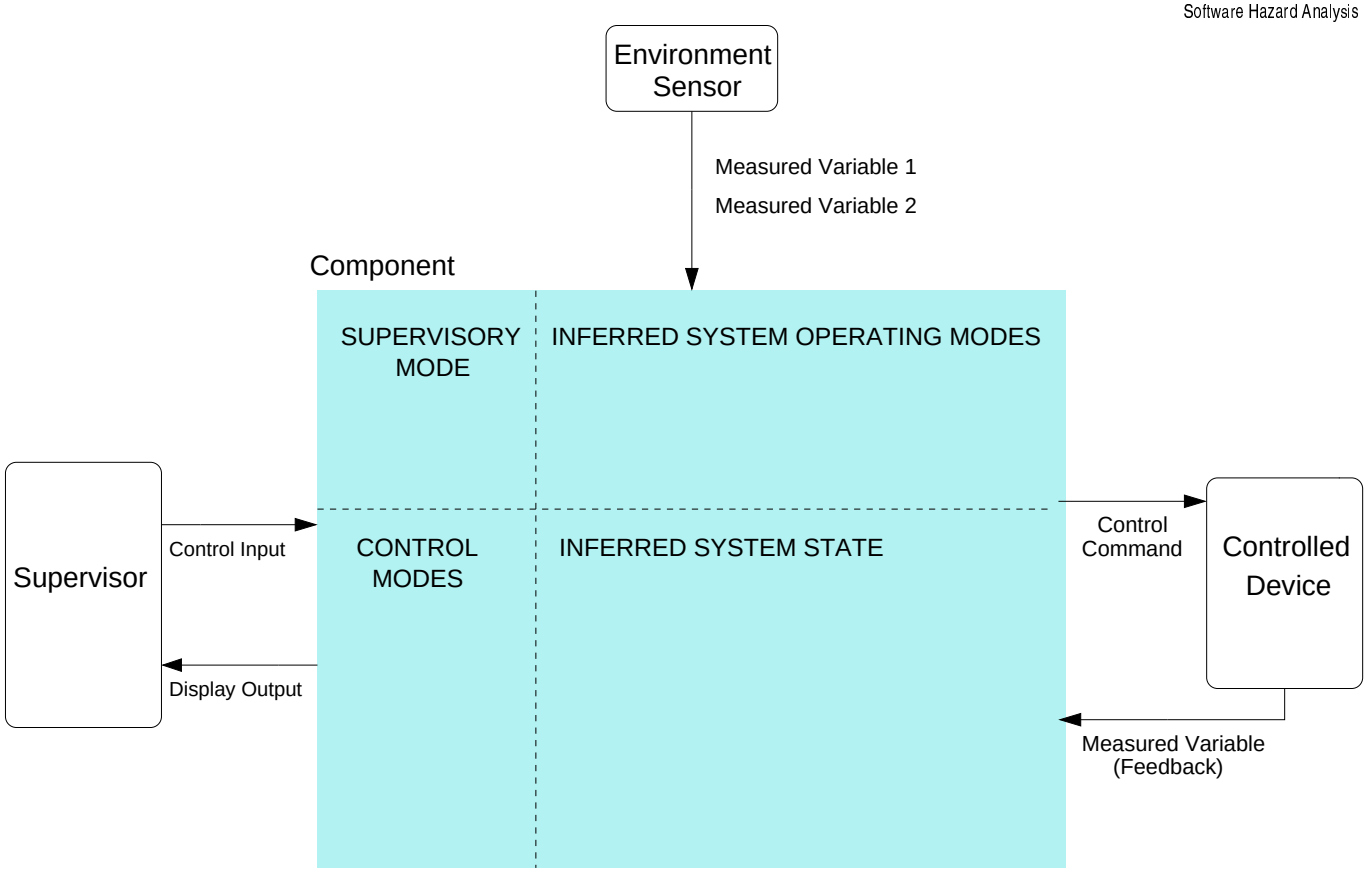
- Combined requirements specification and modeling language
- A state machine with a more readable notation on top of it
  - Includes a task modeling language
  - Could add other notations and visualizations of state machine
- Enforces or includes most of completeness criteria
- Supports specifying systems in terms of modes
  - Control modes
  - Operational modes
  - Supervisory modes
  - Display modes

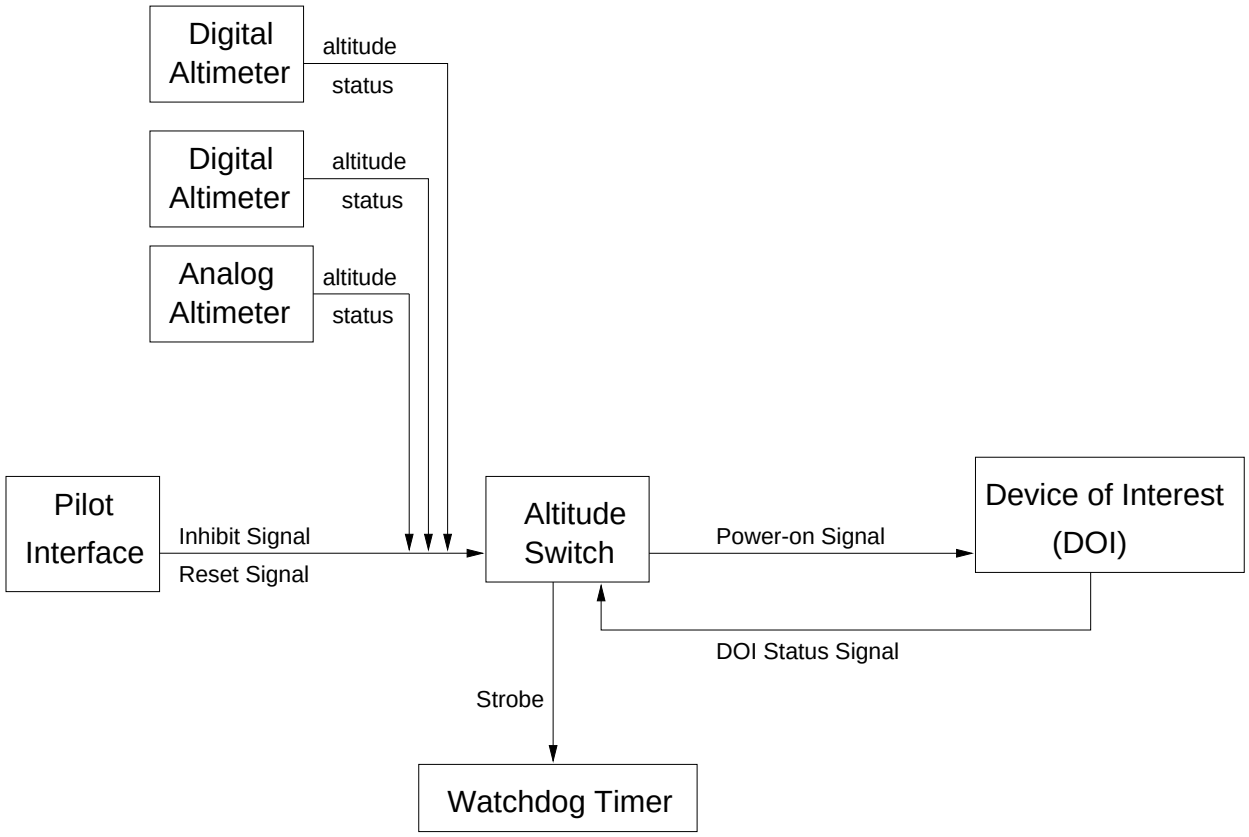
Model of Process

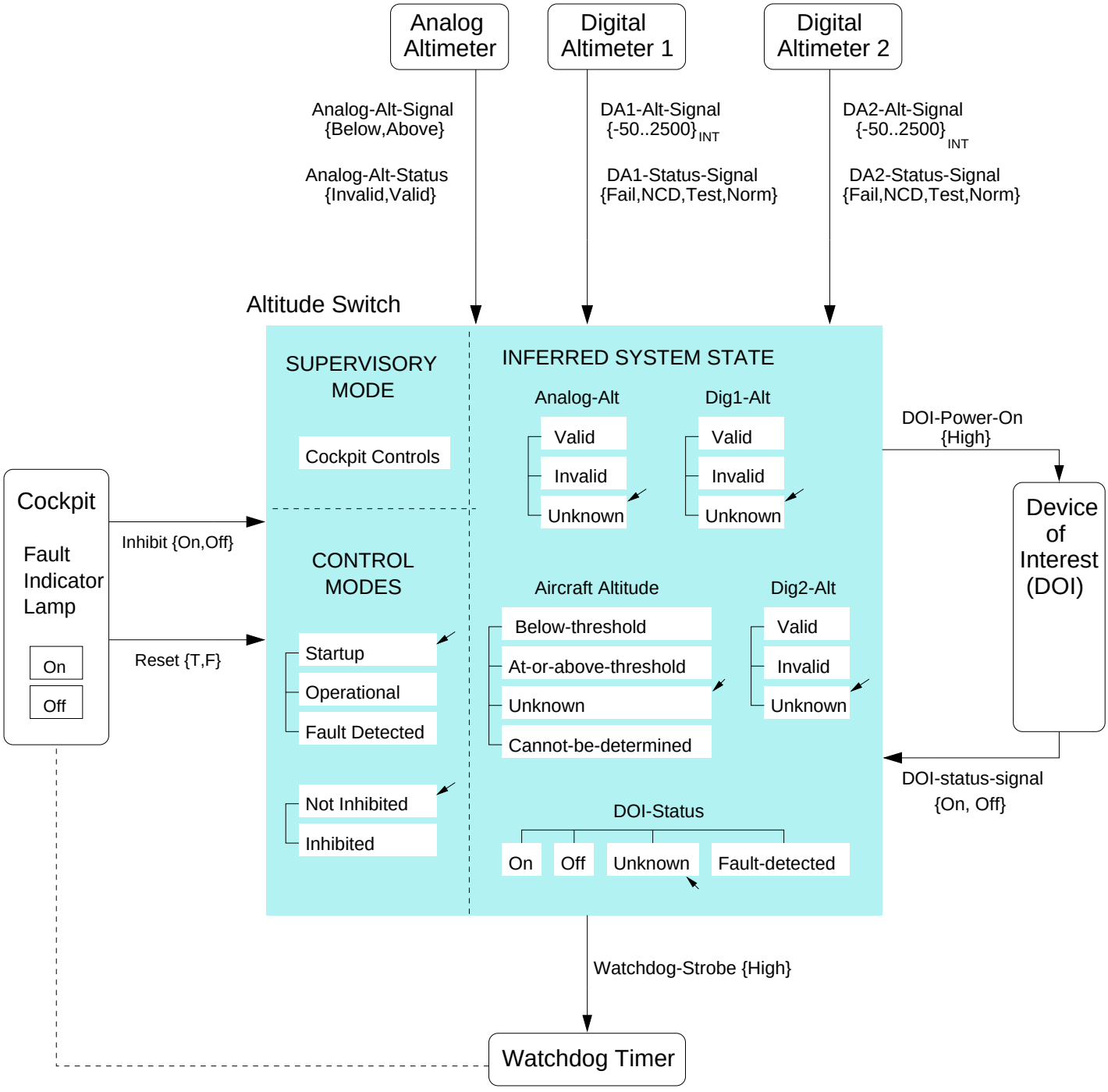
- Process is modeled using state variables



- Values of state variables given by AND/OR tables




Digital Altimeter
altitude
status
Digital Altimeter
altitude
status
Analog Altimeter
altitude
status
Pilot Interface
Inhibit Signal
Reset Signal
Altitude Switch
Power-on Signal
Device of Interest (DOI)
DOI Status Signal
Strobe
Watchdog Timer


Analog Altimeter
Digital Altimeter 1
Digital Altimeter 2
Analog-Alt-Signal {Below,Above}
Analog-Alt-Status {Invalid,Valid}
DA1-Alt-Signal {-50..2500}INT
DA1-Status-Signal {Fail,NCD,Test,Norm}
DA2-Alt-Signal {-50..2500}INT
DA2-Status-Signal {Fail,NCD,Test,Norm}
Altitude Switch
SUPERVISORY MODE
Cockpit Controls
CONTROL MODES
Startup
Operational
Fault Detected
Not Inhibited
Inhibited
INFERRED SYSTEM STATE
Analog-Alt
Valid
Invalid
Unknown
Dig1-Alt
Valid
Invalid
Unknown
Aircraft Altitude
Below-threshold
At-or-above-threshold
Unknown
Cannot-be-determined
Dig2-Alt
Valid
Invalid
Unknown
DOI-Status
On
Off
Unknown
Fault-detected
Cockpit
Fault Indicator Lamp
On
Off
Inhibit {On,Off}
Reset {T,F}
DOI-Power-On {High}
Device of Interest (DOI)
DOI-status-signal {On, Off}
Watchdog-Strobe {High}
Watchdog Timer

Output Command

DOI-Power-On

Destination: DOI

Acceptable Values: {high}

Initiation Delay: 0 milliseconds

Completion Deadline: 50 milliseconds

Exception-Handling: (What to do if cannot issue command within deadline time)

Feedback Information:

Variables: DOI-status-signal

Values: high (on)

Relationship: Should be on if ASW sent signal to turn on

Min. time (latency): 2 seconds

Max. time: 4 seconds

Exception Handling: DOI-Status changed to Fault-Detected

Reversed By: Turned off by some other component or components. Do not know which ones.

Comments: I am assuming that if we do not know if the DOI is on, it is better to turn it on again, i.e., that the reason for the restriction is simply hysteresis and not possible damage to the device.

This product in the family will turn on the DOE only when the aircraft descends below the threshold altitude. Only this page needs to change for a product in the family that is triggered by rising above the threshold.

References: ↑ ↓

CONTENTS

= discrete signal on line PWR set to high

TRIGGERING CONDITION

Control Mode	Operational	T
	Not Inhibited	T
State Values	DOI-Status = On	F
	Altitude = Below-threshhold	T
	Prev(Altitude) = At-or-above-threshold	T

Output Command

Watchdog-Strobe

Destination: Watchdog Timer

Acceptable Values: high signal (on)

Min-Time-Between-Outputs: 0

Max-Time-Between-Outputs: 200_{PERIOD} msec

Exception-Handling:

Feedback Information: None

Reversed By: Not necessary

Comments:

References:

CONTENTS

= High signal on line WDT

TRIGGERING CONDITION

Operating Mode	Operational	T		
	Startup		T	
	Inhibited			T
State Values	Time <= (Time sent Watchdog Strobe) + 200 msec	T		T
	DOI-Status = Fault-detected	F		
	Time >= (Time entered Altitude.Cannot-be-determined) + 2_{DL} secs.	F		

Requirements Analysis

- Model Execution, Animation, and Visualization
- Completeness
- State Machine Hazard Analysis (backwards reachability)
- Software Deviation Analysis
- Human Error Analysis
- Test Coverage Analysis and Test Case Generation

Automatic code generation?

Model Execution and Animation

- SpecTRM−RL models are executable.
- Model execution is animated
- Results of execution could be input into a graphical visualization
- Inputs can come from another model or simulator and output can go into another model or simulator.

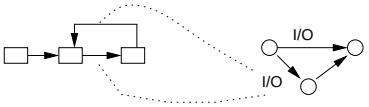
Requirements Completeness

- Most software−related accidents involve software requirements deficiencies.
- Accidents often result from unhandled and unspecified cases.
- We have defined a set of criteria to determine whether a requirements specification is complete.
- Derived from accidents and basic engineering principles.
- Validated (at JPL) and used on industrial projects.

Completeness:	Requirements are sufficient to distinguish the desired behavior of the software from that of any other undesired program that might be designed.

Requirements Completeness Criteria (2)

- How were criteria derived?
  - Mapped the parts of a control loop to a state machine



  - Defined completeness for each part of state machine
    - States, inputs, outputs, transitions
    - Mathematical completeness
  - Added basic engineering principles (e.g., feedback)
  - Added what have learned from accidents

Requirements Completeness Criteria (3)

About 60 criteria in all including human−computer interaction.

(won't go through them all—they are in the book)

Startup, shutdown	Robustness
Mode transitions	Data age
Inputs and outputs	Latency
Value and timing	Feedback
Load and capacity	Reversibility
Environment capacity	Preemption
Failure states and transitions	Path Robustness
Human−computer interface	

Most integrated into SpecTRM−RL language design or simple tools can check them.

Startup and State Completeness

Many accidents involve off−nominal processing modes, including startup and shutdown and handling unexpected inputs.

Examples of completeness criteria in this category:

- The internal software model of the process must be updated to reflect the actual process state at initial startup and after temporary shutdown.
- The maximum time the computer waits before the first input must be specified.
- There must be a response specified for the arrival of an input in any state, including indeterminate states.

Failure States and Transition Criteria

Need to completely specify:

- Off−nominal states and transitions
- Performance degradation
- Communication with operator about fail−safe behavior
- Partial shutdown and restart
- Hysteresis in transitions between off−nominal and nominal

Most accidents occur while in off−nominal processing modes.

Input and Output Variable Completeness

At blackbox interface, only time and value observable to software.

So triggers and outputs must be defined only as constants or as the value of observable events or conditions.

Criteria:

- All information from the sensors should be used somewhere in the specification.
- Legal output values that are never produced should be checked for potential specification incompleteness.

Trigger Event Completeness

- Behavior of computer defined with respect to assumptions about the behavior of the other parts of the system.
- A robust system will detect and respond appropriately to violations of these assumptions (such as unexpected inputs).
- Therefore, robustness of software built from specification will depend on completeness of specification of environmental assumptions.
  - There should be no observable events that leave the program's behavior indeterminate.
- Why need to document and check all assumptions?

Formal Robustness Criteria

- To be robust, the events that trigger state changes must satisfy the following:
  1. Every state must have a behavior (transition) defined for possible input.
  2. The logical OR of the conditions on every transition out of every state must form a tautology.

     $x < 5$

     $x \geq 5$
  3. Every state must have a software behavior (transition) defined in case there is no input for a given period of time (a timeout).
- Together these criteria guarantee handing input that are within range, out of range, and missing.

Nondeterminism Criterion

- The behavior of the requirements should be deterministic (only one possible transition out of a state is applicable at any time).

  $X > 0$

  $X < 2$

- We (and others) have tools to check specifications based on state machines for robustness, consistency, and nondeterminism.

NOTE: This type of mathematical completeness is NOT enough.

e.g., *"true"* is a mathematically complete, consistent, and deterministic specification.

Value and Timing Assumptions

Examples:

- All inputs should be checked and a response specified in the event of an out−of−range or unexpected value.
- All inputs must be fully bounded in time and the proper behavior specified in case the limits are violated.
- Minimum and maximum load assumptions ...
- A minimum−arrival−rate check should be required for each physically distinct communication path.

  Software should have the capability to query its environment with respect to inactivity over a given communication path.

- Response to excessive inputs (violations of load assumptions) must be specified.

Human−Computer Interface Criteria

- For every data item displayable to a human, must specify:
    1. What events cause this item to be displayed?
    2. What events cause item to be updated?
       If so, what events should cause the update?
    3. What events should cause the display to disappear?
- For queues need to specify:
    1. Events to be queued
    2. Type and number of queues to be provided (alert and routine)
    3. Ordering scheme within queue (priority vs. time of arrival)
    4. Operator notification mechanism for items inserted in the queue.
    5. Operator review and disposal commands for queue entries.
    6. Queue entry deletion.

Environment Capacity Constraints

Examples:

- For the largest interval in which both input and output loads are assumed and specified, the absorption rate of the output environment must equal or exceed the input arrival rate.

- Contingency action must be specified when the output absorption rate limit will be exceeded.

Data Age Criteria

- All inputs used in specifying output events must be properly limited in the time they can be used.
- Output commands that may not be able to be executed immediately must be limited in the time they are valid.
- Incomplete hazardous action sequences (transactions) should have a finite time specified after which the software should be required to cancel the sequence automatically and inform the operator.
- Revocation of partially completed transactions may require:
  1. Specification of multiple times and conditions under which varying automatic cancellation or postponement actions are taken without operator confirmation.
  2. Specification of operator warnings to be issued in case of such revocation.

Latency Criteria

- Latency is the time interval during which receipt of new information cannot change an output even though it arrives prior to output.
  - Influenced by hardware and software design (e.g., interrupt vs. polling)
  - Cannot be eliminated completely.
  - Acceptable length determined by controlled process.
- Subtle problems when considering latency of HCI data.
  (see book for criteria)

Feedback Criteria

Basic feedback loops, as defined by the process control function, must be included in the requirements along with appropriate checks to detect internal or external failures or errors.

Examples:

- There should be an input that the software can use to detect the effect of any output on the process.
- Every output to which a detectable input is expected must have associated with it:
    1. A requirement to handle the normal response
    2. Requirements to handle a response that is missing, too late, too early, or has an unexpected value.

Path Criteria

- Paths between states are uniquely defined by the sequence of trigger events along the path.
- Transitions between modes are especially hazardous and susceptible to incomplete specification.

REACHABILITY

- Required states must be reachable from initial state.
- Hazardous states must not be reachable.
- Complete reachability analysis often impractical, but may be able to reduce search by focusing on a few properties or using backward search.
- Sometimes what is not practical in general case is practical in specific cases.

Path Criteria (2)

RECURRENT BEHAVIOR

- Most process control software is cyclic. May have some non−cyclic states (mode change, shutdown)
- Required sequences of events must be specified in and limited by transitions in a cycle.
- *Inhibiting state:* State from which output cannot be generated.
- There should be no states that inhibit later required outputs.

REVERSIBILITY

PREEMPTION

Path Criteria (3)

PATH ROBUSTNESS

Soft failure mode: The loss of ability to receive input X could inhibit the production of output Y

Hard failure mode: The loss of ability to receive input X will inhibit the production of output Y

- Soft and hard failure modes should be eliminated for all hazard reducing outputs.
- Hazard increasing outputs should have both soft and hard failure modes.
- Multiple paths should be provided for state changes that maintain safety.
- Multiple inputs or triggers should be required for paths from safe to hazardous states.

CONSTRAINT ANALYSIS

State Machine Hazard Analysis

- Start from a hazardous configuration in the model (violates safety design constraint)
- Trace backward until get enough information to eliminate it from design.
- Natasha Neogi has extended to hybrid models.

Human Error Analysis

- General requirements and design criteria
- Hazard analysis for specific hazards
- Mode Confusion and other Analysis
  - Want to look at interaction between human controllers and the computer
  - Have designed an operator task modeling language using same underlying formal model.
  - Can be executed and analyzed along with other parts of model.

Operator Task Models

- To ensure safe and efficient operations, must look at the interaction between the human controllers and the computer.
- Use same underlying formal modeling language.
- Designed a visual representation more appropriate for the task modeling.
- Can be executed and analyzed along with other parts of the model.

Executable Specifications as Prototypes

- Easily changed
- At end, have specification to use
- Can be reused (product families)
- Can be more easily reviewed
- If formal, can be analyzed
- Can be used in hardware−in−the−loop or operator−in−the−loop simulations