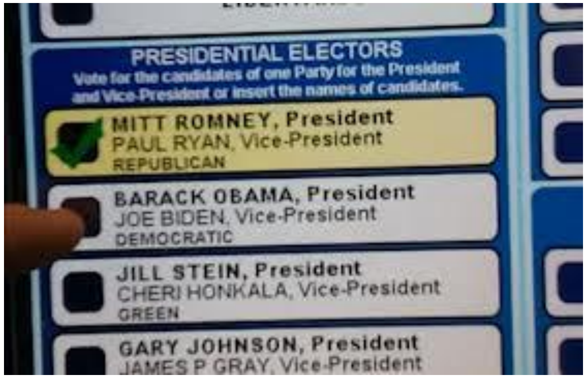
MODELING AND SPECIFYING REQUIREMENTS FOR SAFETY-CRITICAL SYSTEMS

Connie Heitmeyer
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC

Software Cerification Consortium Workshop
NRC, Rockville, MD
Oct. 28 – Oct. 29, 2013

THE CURRENT STATE OF SOFTWARE

SOFTWARE IS EVERYWHERE



Dramatic Increase in the **Software Size**

There has been a huge increase in the amount of software in many industrial systems, e.g., automobiles

- In 1981, General Motors passenger cars executed ~50 KLOC
- Today's *average car* contains more than 1 MLOC
- Today's *premium class car* is estimated to contain in excess of 100 MLOC!

Increase in Code Size: Factor of 20-2000!

Dramatic Increase in **Software Complexity**

Software in Military Aircraft

Year	Aircraft	% of Pilot Functions
1960	F-4	8%
1982	F-16	45%
2000	F-22	80%
In Testing	F-35	90%

Automotive Functions Supported by Today's Software

Air Bag System	Antilock Brakes	Automatic Transmission
Alarm System	Climate Control	Collision Avoidance
Cruise Control	Communication System	Dashboard Instrum.
Electronic Stability Control	Engine Ignition	**Engine Control**
Electronic Seat Control	Entertainment System	Navigation
Power Steering	Tire Pressure Monitoring	Windshield Wiper Control

THE SOFTWARE PROBLEM

SOFTWARE FAILURES IN DEFENSE SYSTEMS

Due to a Software Problem, Navy Drone Wanders Into Restricted Airspace Near Washington

Result: **Grounding of all six of Navy's Fire Scouts**

…."When contact with the Fire Scout is lost, a program is supposed to have it immediately return to the airfield to land safely. That did not happen as planned."

New York Times, Aug 25, 2010

A U.S. soldier in Afghanistan used a Precision Lightweight GPS Receiver to set coordinates for an air strike. Seeing that the "battery low" warning light was on, he changed the battery, then pressed "Fire." The device was designed, on starting or resuming operation after a battery change, to initialize the coordinate variables to its own location…

The soldier and three comrades were killed in the incident.

" 'Friendly Fire' Deaths Traced to Dead Battery: Taliban Targeted, but US Forces Killed," *Wash. Post*, 22 Mar. 2002

GROWING CONCERNS ABOUT THE SAFETY OF AUTONOMOUS SYSTEMS

In DoD, many unmanned vehicles already deployed and others under development (robots, UAVs, UGVs, …)

- Larger and faster
- Greater complexity in functions, environments
- DoD estimated to deploy over 7,000 UAVs currently compared to less than 50 a decade ago
- More opportunities for serious safety violations
  - Recent incident: UGV drags IED toward Ordinance Disposal Team

Plans exist to deploy unmanned systems in non-military applications, such as law enforcement and public safety

- Law enforcement: Equip UAVs with cameras and scientific instruments for surveillance and information gathering and with weapons, such as rubber bullets, Tasers, and tear gas

BENEFITS OF FORMAL REQUIREMENTS MODELS

REQUIREMENTS MODELS

- Purpose of a **system requirements model**
  - Specifies the set of all acceptable implementations
  - Avoids overspecification (e.g., implementation bias)
    - Excludes acceptable implementations
  - Avoids underspecification (incompleteness)
    - Allows unacceptable implementations
- Components of a req. model (Parnas 4-Var Model)
  - The required relation among entities in the system environment, monitored and controlled entities (REQ)
    - In response to changes in the values of monitored quantities, system changes values of controlled quantities
  - Environmental assumptions that constrain the values of the monitored and controlled quantities (NAT)
    - physical laws and constraints imposed by the system environment
- Dual-Language Approach
  - Operational spec
  - Property-based spec

FORMAL MODELING CAN EXPOSE ERRORS IN MORE INFORMAL MODELS

- System: FDIR module in software for Intern. Space Stn.
- System purpose: If failure occurs, output failure notification and/or sound one of two different alarms
- Available resources: Domain expert + two req. documents
- Results
  - Formulating a formal model of the required software behavior exposed two serious errors *in less than one week's time*
    - The action required in two modes had been erroneously switched
    - The spec contained undesirable implementation bias

FDIR: Failure Detection, Isolation & Recovery
in Internat. Space Station's Thermal Radiator Rotary software

ANALYZING FORMAL MODELS CAN EXPOSE WELL-FORMEDNESS ERRORS

Check that functions are total (no missing cases) and well-defined (deterministic behavior)

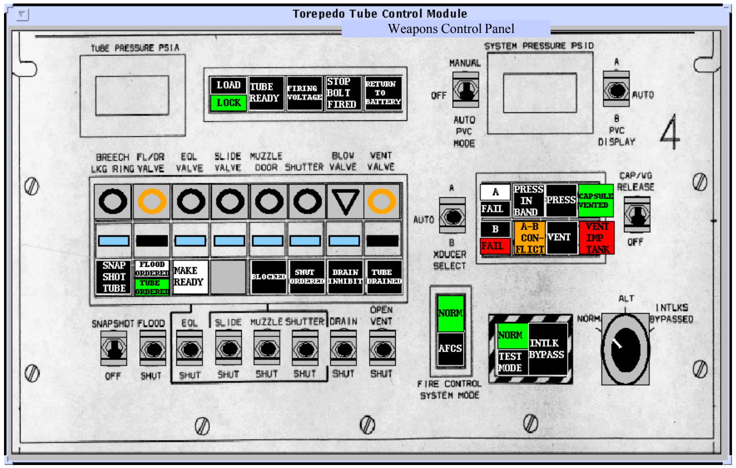
- Checked software req. document for Navy **aircraft's Flight Program**
  - Had been checked manually for errors by two independent review teams
- Results
  - Check of 36 function definitions
    - Detected 17 missing cases
  - Checked a total of 4319 logical expressions defining mode transitions
    - Detected 57 instances of non-determinism

Example: Input that could trigger transition from Inertial mode to either Doppler_Inertial or Air_Alignment mode

```
Doppler_up' WHEN [NOT CA_stage_complete AND
latitude > 70 deg. AND NOT present_position_entered
AND NOT latitude > 80 deg. AND IMSMODE=Gndal]
```

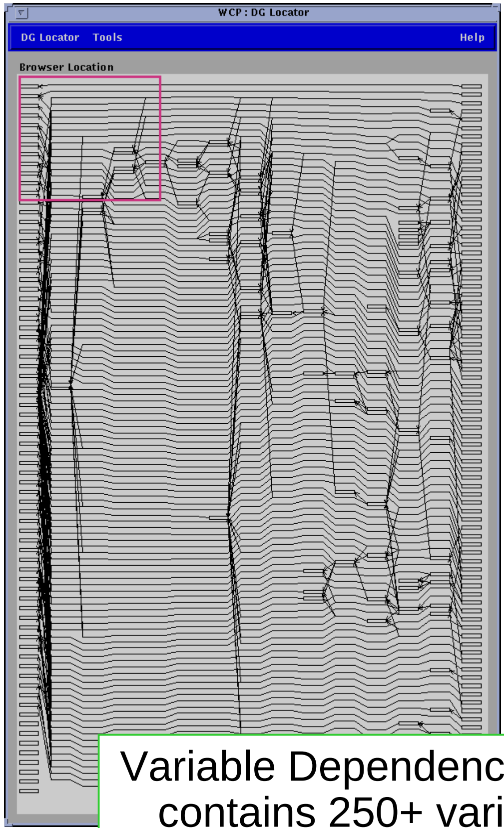
"Consistency checking" finds MANY errors that human inspections miss and does so very quickly (seconds to minutes)

ANALYZING FORMAL REQ. MODELS CAN DETECT SAFETY VIOLATIONS

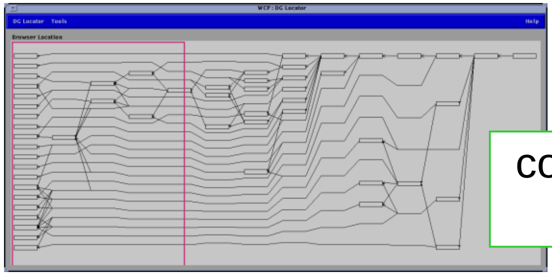


Weapons Control Panel

- Used to monitor the status, prepare launch of weapons
- Sizable, complex program (~30KLOC)
- Contractor software requirements spec contains 250+ variables



Variable Dependency Graph contains 250+ variables



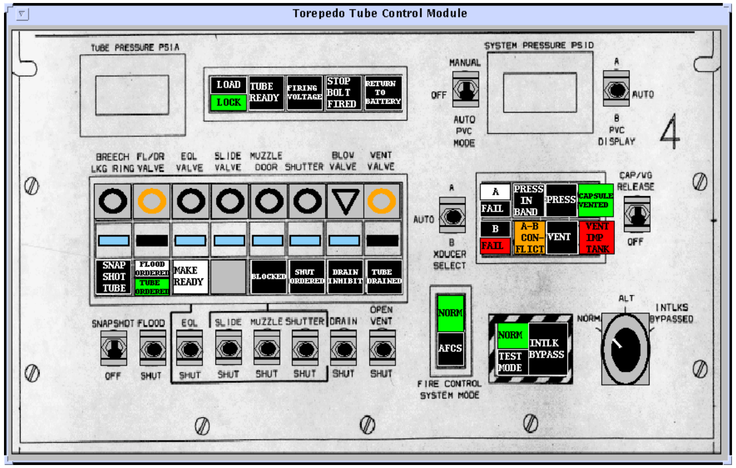
contains 55 variables (~80% reduction)

- Analyzed for six safety properties
- Original spec too large to analyze
  - too many variables
  - several real-valued vars
- Applied abstractions
  - Slicing
  - Data abstraction

AN EXECUTABLE FORMAL MODEL CAN BE USED FOR VALIDATION

Validating the model

- Because our model is executable, we can “simulate” the system behavior
- Used a GUI builder to build a realistic front-in
- Domain experts can run the simulator to validate that the model captures the intended behavior

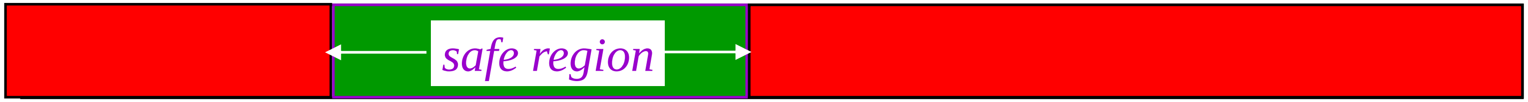


Checking for spurious property violations

- Because our data abstractions were not complete, we needed to check for spurious safety violations: Insure that each violation is reachable
- To do so, we ran the counterexamples returned by model checking through the simulator

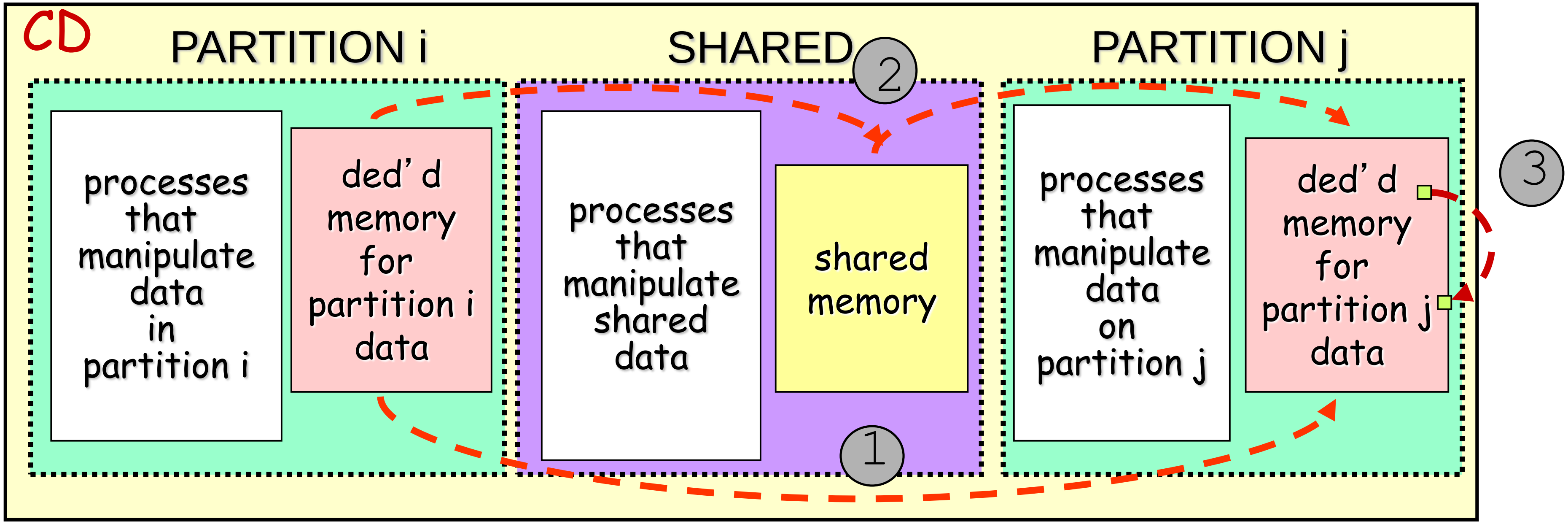
One of safety properties →

Opening the Torpedo Tube Vent Valve shall be prevented unless the Missile-to-Torpedo-Tube differential pressure is within safe limits



WE CAN PROVE THAT FORMAL REQ. MODELS SATISFY SECURITY PROPERTIES

- CD: Embedded software that processes data in diff. memory areas
- Data in diff. areas may be classified at diff. security levels
- Required security property: **DATA SEPARATION**
  - Data or activity in one area cannot influence or be influenced by data or activity in another area



- To support a EAL6+ Common Criteria certification, delivered formal model, sets of security properties & assumptions, proof that model satisfies properties, annotated C code, and demo that C code refined model

THE REQUIREMENTS PROBLEM

PROBLEM:
WRITING GOOD REQUIREMENTS IS HARD!

The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements…No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

Fred Brooks
"No Silver Bullet: Essence and Accidents of Software Eng.," IEEE Computer, 1987

THE REQUIREMENTS PROBLEM*

In spite of…advancements…, biggest problem in software engineering [is] bridging of ‘gap’ between the intent captured in requirements and expressed at a high level, and the detailed encoding of this intent in code.

Sriram Rajamani, "Software is more than code"

A final difficulty encountered in modeling is the frequent lack of good requirement documents associated with the project. Most of the time, industrial requirement documents are either almost nonexistent or far too verbose. Usually they have to be rewritten before serious modeling starts.

Jean-Raymond Abrial, "Theory becoming practice"

There is general consensus that the most significant problems in software development are due to inadequate requirements, especially where these concern what one component or subsystem may expect of another.

John Rushby, "Automated formal methods enter the mainstream"

**Journal of Universal Computer Science, May 2007*

FORMAL MODELING/ANALYSIS OF CD FOR SECURITY PROPERTIES

Requirements Acquisition	years…
Formulate the Requirements Model & the Security Properties	2.5+ weeks
Translate the Req. Model to Lang. of Thm Prover & Construct the Proofs	3+ weeks
Demonstrate Code Conformance	5+ weeks
Annotate the Code	many months…

HOW TO DEVELOP A SYSTEM REQUIREMENTS MODEL FOR LARGE, COMPLEX SOFTWARE SYSTEMS

WE NEED GOOD SYSTEM ABSTRACTIONS

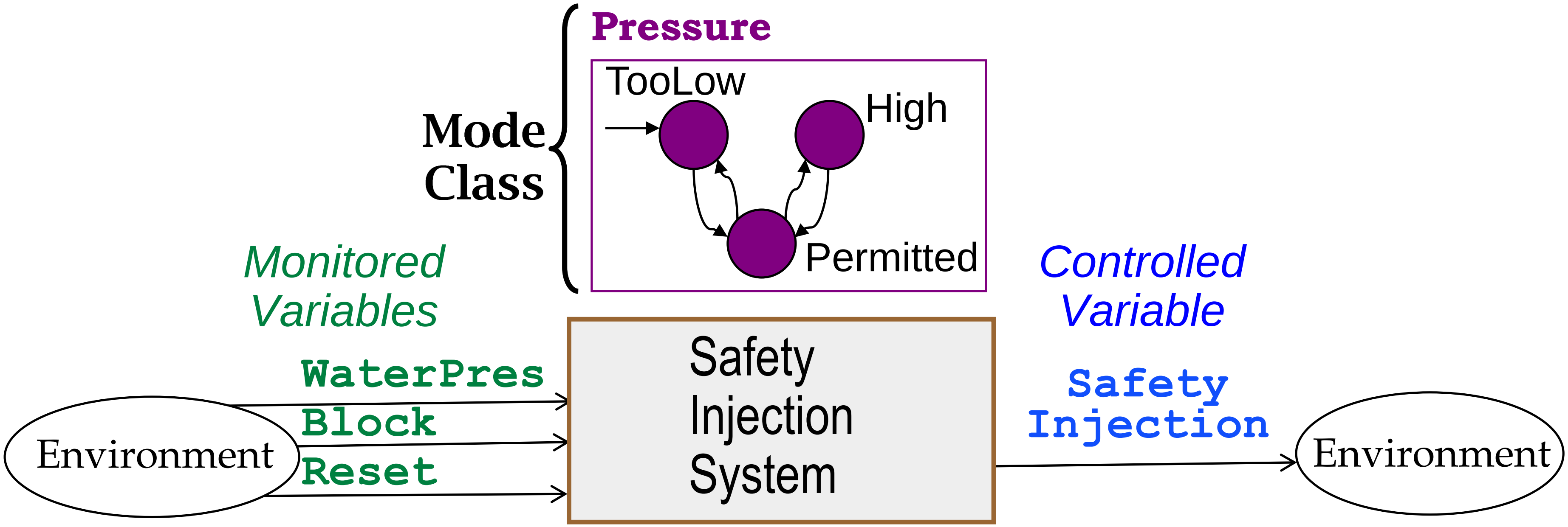
ADVANTAGES

- Facilitate “divide and conquer”
- Allow “separation of concerns”
- Make large models
  - easier to understand
  - easier to reason about
  - easier to change
- Facilitate incremental development
- Provide a solid foundation for program families (i.e., product lines)

EXAMPLE OF A GOOD SYSTEM-LEVEL ABSTRACTION: **MODES**

A **mode class** is a set of system modes

- Partitions the system state into equivalence classes
- When the system is one mode, its behavior is significantly different than when it is in a different mode
- **Many systems have more than one mode class**



System controlling safety injection in a nuclear plant

MODES ARE ALREADY PRESENT IN THE SPECS OF MANY SYSTEMS

Operational Flight Program for Navy Aircraft

Ardupilot UAV

1	2	3	4	5	6	7	8
ID	**Failure Condition**	**Failure Detection Phase**	**Failure Criteria**	**Persistence Time**	**Failure Notifications**	**Recovery Response**	**Inhibit**
1a	Failure to Autotrack: response not inhibited	Autotrack Mode	Position_Err $\geq$ Autotrack_Error	Pers_Autotrack_Failure	Autotrack_Failure, Joint_Failure	Transition to Switchover Mode	Inhibit_String
1b	Failure to Autotrack: response inhibited	Autotrack Mode	Position_Err $\geq$ Autotrack_Error	Pers_Autotrack_Failure	Autotrack_Failure, Joint_Failure	Device_Power_Off, Transition to Checkout Mode	Inhibit_String
...	...	...	...	...	...	...	...
5	Blind Ops timeout exceeded	Blind Mode and Torque Motor On	Blind duration > Limit + 1	None	Time_Limit_Blind	Transition to Shutdown Mode	Inhibit_Blind
...	...	...	...	...	...	...	...
7	String failure: response inhibited not	Autotrack Mode	Receive CWA_String_Failure	Pers_String_Failure	Joint_Failure	Transition to Switchover Mode	Inhibit_String
8	String failure: response inhibited	Autotrack Mode	Receive CWA_String_Failure	Pers_String_Failure	Joint_Failure	Device_Power_Off, Transition to Checkout Mode	Inhibit_String
	...	...	...	...	...	...	...

Failure, Detection, Isolation & Recovery in the Thermal Radiator Rotary Joint Manager NASA Software for the International Space Station

FACILITATING THE DEVELOPMENT OF FORMAL REQUIREMENTS MODELS USING MODES

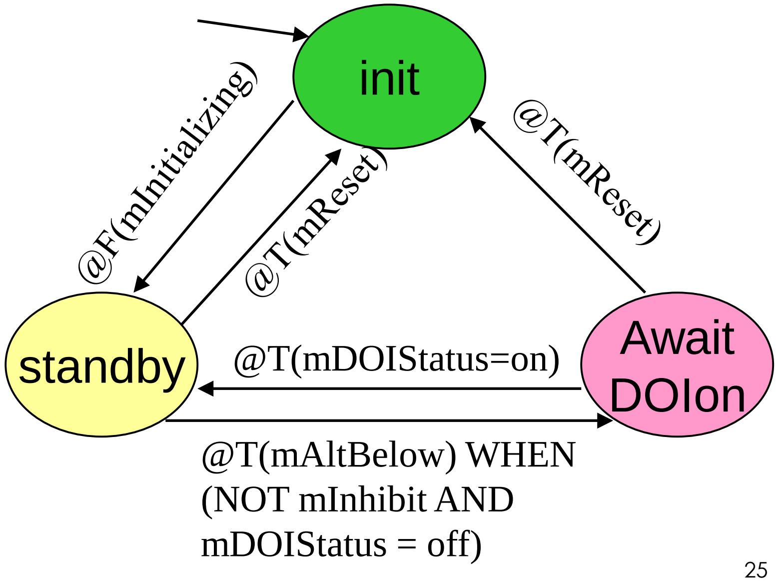
USING FORMAL MODELS AND MODES IN INCREMENTAL DEVELOPMENT

Real-World Avionics System: Altitude Switch (ASW)

- Normal System Behavior ID
  - Controller powers on a generic Device of Interest (DOI) when the aircraft descends below a threshold altitude
  - Pilot sets an inhibitor button to prevent powering on of DOI
  - Pilot presses a reset button to reinitialize the ASW

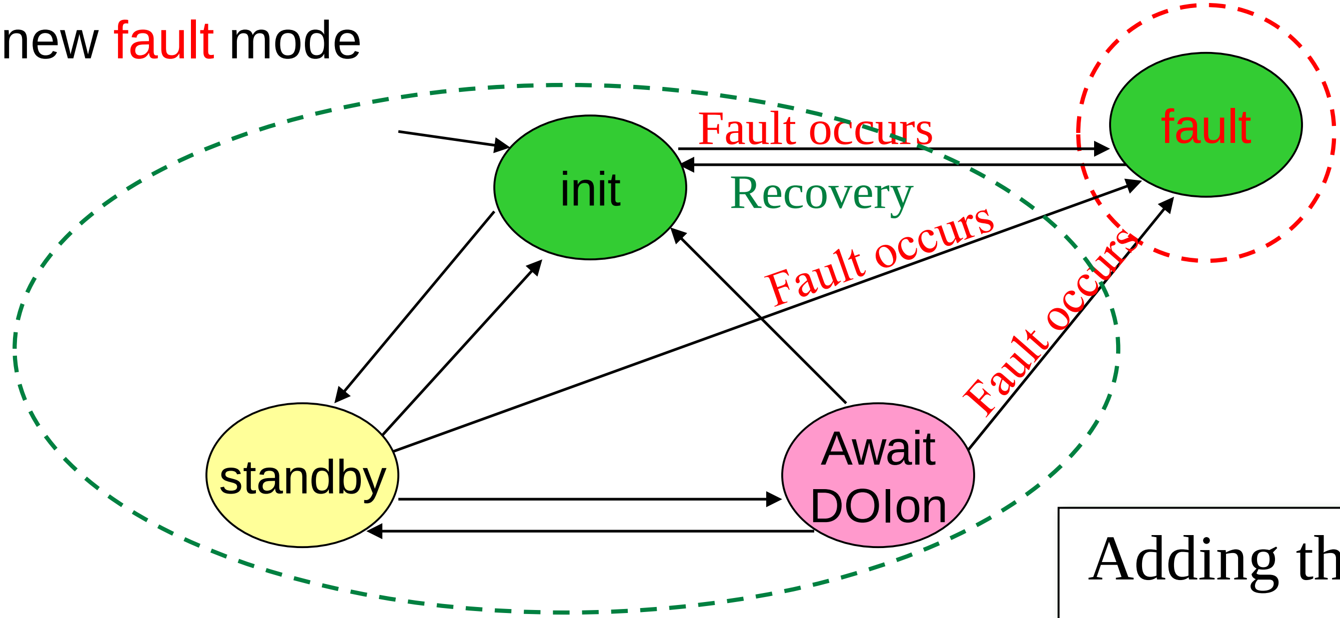
- Fault-Tolerant Syst. Behavior FT
  - When a fault occurs (e.g., no input
  - within given time interval) system enters fault mode & turns on a fault indicator light
  - System recovers when the pilot hits reset

Mode Transitions of ID



CONSTRUCTING THE FAULT-TOLERANT MODEL FROM THE NORMAL MODEL

- Add new values to value set of variables of ID
  - E.g. new fault mode



Adding these three components to the normal model ID defines the fault-tolerant model FT

- Add new fault-tolerance variables
  - mAltimeterfail: Failure of all three altimeters
  - mTime: Discrete time given by system clock
  - cFaultIndicator: On iff system is in fault mode
- Add new transitions for fault handling and fault recovery

ADVANTAGES OF DEVELOPING FAULT-TOLERANT MODEL INCREMENTALLY

- A divide and conquer approach breaks the problem into smaller subproblems
- Separation of concerns – define normal behavior first and fault-tolerant behavior later
- Proving properties of FT is facilitated by
  - Property inheritance rules
  - Compositional proof rules
- Many properties of FT can be proven automatically from properties proven about ID!

USING FORMAL MODELS AND MODES IN DEVELOPING PRODUCT LINES

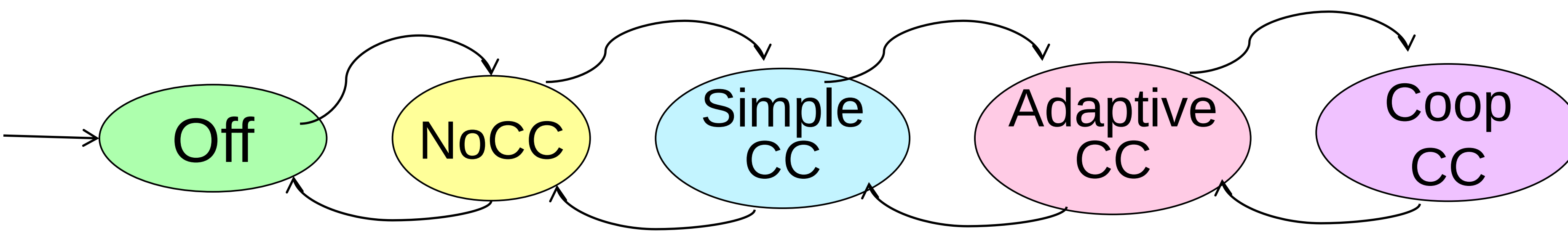
CRUISE CONTROL

Different cruise control modes

- Manual driving
- Simple cruise control
- Adaptive: Maintain min dist from car in front
- Cooperative: Communicate with car in front

Approach

- Develop each new version incrementally from the previous version



USING FORMAL MODELS AND MODES: INCREMENTAL DEV. USING COMPOSITION

Model of a software controller of a UAV called ArduPilot (AP)

- While in flight, AP is in one of six navigation modes
- The navigation mode determines how thrust, pitch and throttle are computed.

	Current Mode mcNav	Event	New Mode
1	Manual, Stabilize, Auto, Fly-by-wire, Loiter	@T(mLow-battery) OR @T(mSwitch-pos=rtl)	RTL
2	Auto	@T(mReached-waypt) WHEN tLast-waypt	RTL
3	Manual, Fly-by-wire, Stabilize	@T(mFailsafe)	RTL
4	RTL	@T(mSwitch-pos = loiter) OR @T(mReached-launch-site)	Loiter
5	Manual, Fly-by-wire, Stabilize, Auto	@T(mSwitch-pos = loiter)	Loiter
6	Stabilize, Auto, Fly-by-wire, RTL, Loiter	@T(mSwitch-pos = manual)	Manual
7	Manual, Fly-by-wire, Auto, RTL, Loiter	@T(mSwitch-pos = stabilize)	Stabilize
8	Manual, Stabilize, Auto, RTL, Loiter	@T(mSwitch-pos = fly-by-wire)	Fly-by-wire
9	Manual, Stabilize, Loiter, Fly-by-wire, RTL	@T(mSwitch-pos = auto)	Auto

ArduPilot **Mode Transitions** for Navigation

	Current Mode mcNav	cDesiredRoll
1	Stabilize	mActualRoll
2	Fly-by-wire	mDesiredRoll
3	Auto, RTL, Loiter	F1(mActualLoc, tNextWP, mActualRoll, mHead1, mHead2)
4	Manual	mActualRoll

Function for computing cDesired Roll

Functions computing cDesiredPitch and cDesiredThrottle may be similarly defined

USING FORMAL MODELS AND MODES: INCREMENTAL DEV. USING COMPOSITION

- Suppose ArduPilot is equipped with a video camera.
- When ArduPilot is in flight, a ground operator could command the controller to switch the camera on, take video of a designated area, and transmit the video to some location.

	Current Mode mcCam	Event	New Mode
1	V-Off	@T(mSwitch-video=on)	V-On
2	V-On	@T(mSwitch-video=off)	V-Off
3	V-On	@T(mStart-video)	Video-in-Progress
4	Video-in-Progress	@T(mStop-video)	V-On

Mode Transitions for Camera

	Current Mode mcCam	cXmtVideo
1	V-Off, V-On	Off
2	Video-in-Progress	On

Function defining cXmtVideo

USING FORMAL MODELS AND MODES: INCREMENTAL DEV. USING COMPOSITION

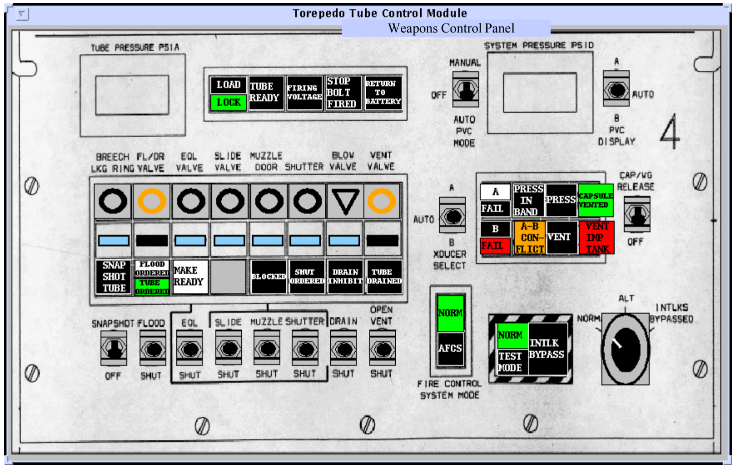
- Assume that, in Stabilize mode, the camera cannot take video but that in all other navigation modes, it can.
- Composing the two models produces a new composite model made up of
  - The original ArduPilot model which only performed navigation
  - The original Camera model with a modified mode transition table
  - The functions defining throttle, pitch, roll and XmtVideo are unchanged
- This is parallel composition but the "feature interaction" problem needs to be addressed.
  - If a is a mode in mode class A and b is a mode in mode class B, we may need to specify that when the system is in a, it cannot be in b

	Current Mode mcCam	Event	New Mode
1	V-Off	@T(mSwitch-video=on)	V-On
2	V-On	@T(mSwitch-video=off)	V-On
3	V-On	@T(mStart-video) WHEN mcNav ≠ Stabilize	Video-in-Progress
4	Video-in-Progress	@T(mStop-video) OR @T(mcNav = Stabilize)	V-On

New Mode Transitions for Camera

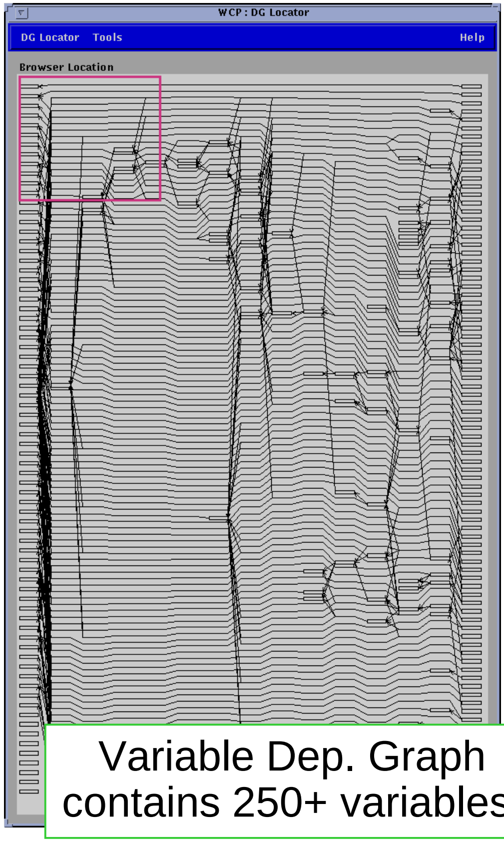
REQUIREMENTS SPECS IN INDUSTRY

SOFTWARE REQUIREMENTS SPEC PRODUCED BY A NAVY CONTRACTOR

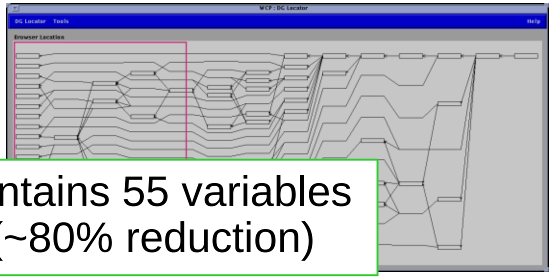


Weapons Control Panel

- Used to monitor the status, prepare launch of weapons
- Sizable, complex program (~30KLOC)
- Contractor software requirements spec contains 250+ variables



Variable Dep. Graph contains 250+ variables



contains 55 variables (~80% reduction)

- Requirements expressed as 'logic equations'
- Semi-automatic translation to our tabular notation
  - Took < one week
- Included six safety properties
- Included pictures of operator interface
  - Used to develop a graphical front-end for simulator
  - Navy personnel could use our simulator to validate the spec

EXCERPT FROM NASA'S ORIGINAL REQUIREMENTS DOCUMENT FOR FDIR

1	2	3	4	5	6	7	8
ID	**Failure Condition**	**Failure Detection Phase**	**Failure Criteria**	**Persistence Time**	**Failure Notifications**	**Recovery Response**	**Inhibit**
1a	Failure to Autotrack: response not inhibited	Autotrack Mode	Position_Err $\geq$ Autotrack_Error	Pers_Autotrack _Failure	Autotrack_ Failure, Joint_ Failure	Transition to Switchover Mode	Inhibit_ String
1b	Failure to Autotrack: response inhibited	Autotrack Mode	Position_Err $\geq$ Autotrack_Error	Pers_Autotrack _Failure	Autotrack_ Failure, Joint_ Failure	Device_ Power_Off, Transition to Checkout Mode	Inhibit_ String
	...	...	...	...	...	...	...
5	Blind Ops timeout exceeded	Blind Mode and Torque Motor On	Blind duration $>$ Limit + 1	None	Time_Limit_ Blind	Transition to Shutdown Mode	Inhibit_ Blind
	...	...	...	...	...	...	...
7	String failure: response inhibited not	Autotrack Mode	Receive CWA_Str ing_Failure	Pers_String _Failure	Joint_ Failure	Transition to Switchover Mode	Inhibit_ String
8	String failure: response inhibited	Autotrack Mode	Receive CWA_Str ing_Failure	Pers_String _Failure	Joint_ _Failure	Device_ Power_Off, Transition to Checkout Mode	Inhibit_ String
	...	...	...	...	...	...	...

FDIR: Failure Detection, Isolation & Recovery

in Internat. Space Station's Thermal Radiator Rotary software

TRANSLATION FROM FDIR TABLE TO MODE TRANS. TABLE IS EASY

1	2	3	4	5	6	7	8
ID	Failure Condition	Failure Detection Phase	Failure Criteria	Persistence Time	Failure Notifications	Recovery Response	Inhibit
1a	Failure to Autotrack: response not inhibited	Autotrack Mode	Position_Err ≥ Autotrack_Error	Pers_Autotrack_Failure	Autotrack_Failure, Joint_Failure	Transition to Switchover Mode	Inhibit_String
1b	Failure to Autotrack: response inhibited	Autotrack Mode	Position_Err ≥ Autotrack_Error	Pers_Autotrack_Failure	Autotrack_Failure, Joint_Failure	Device_Power_Off, Transition to Checkout Mode	Inhibit_String
	...	...	...	...	...	...	...
5	Blind Ops timeout exceeded	Blind Mode and Torque Motor On	Blind duration > Limit + 1	None	Time_Limit_Blind	Transition to Shutdown Mode	Inhibit_Blind
	...	...	...	...	...	...	...

Current Mode	Events	New Mode
Blind	@T(mBlind_Timeout) WHEN (mTorque_Motor_On AND NOT mInhibit_Blind)	Shutdown
Autotrack	@T(tPers_Autotrack_Fail) WHEN (NOT Inhibit_String)	Switchover
Autotrack	@T(tPers_Autotrack_Fail) WHEN Inhibit_String	Checkout
...	...	...

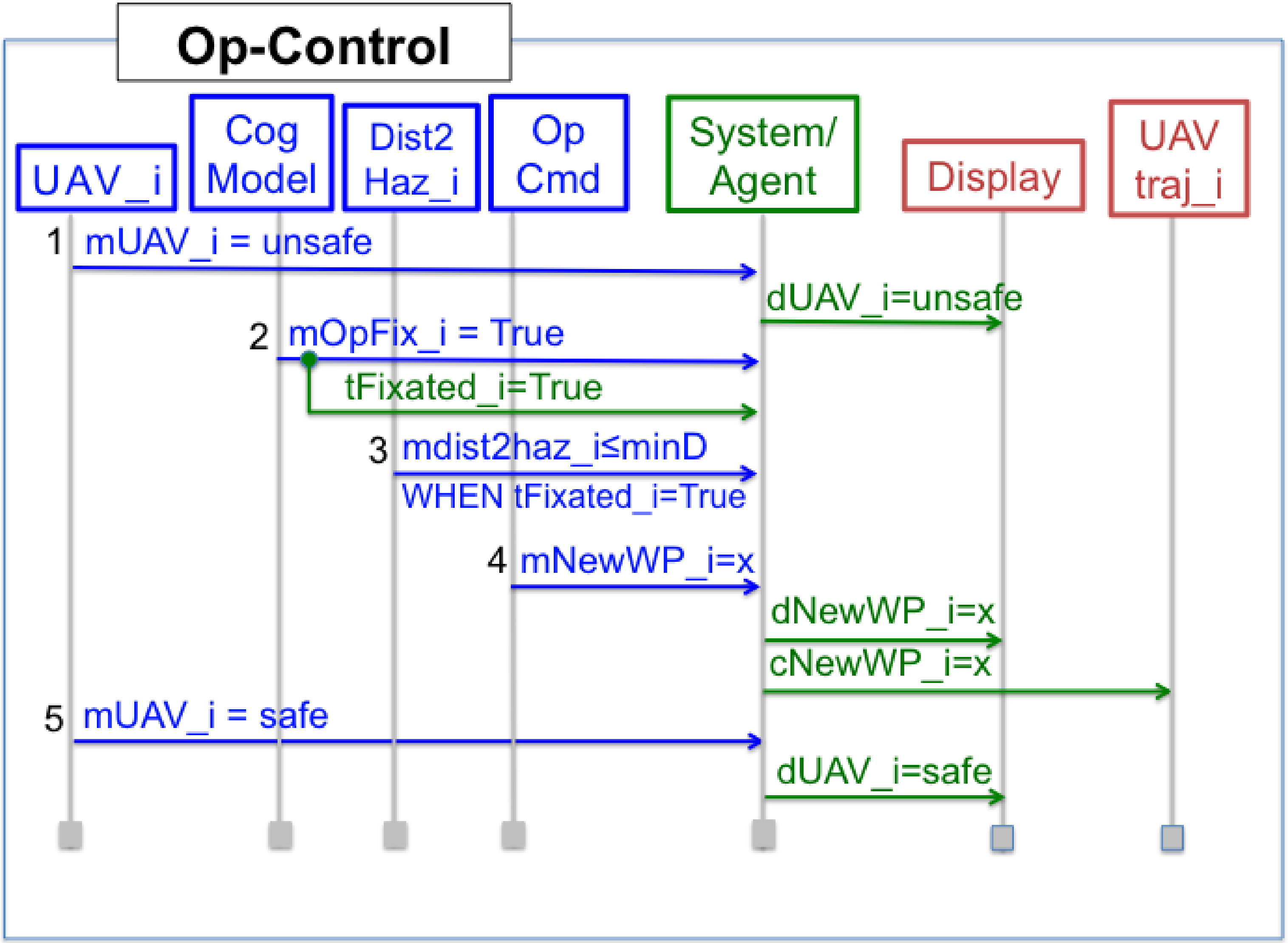
Excerpt from mode transition table for FDIR

SYNTHESIS OF FORMAL SYSTEM REQUIREMENTS MODELS

REQUIREMENTS MODELS IN INDUSTRY

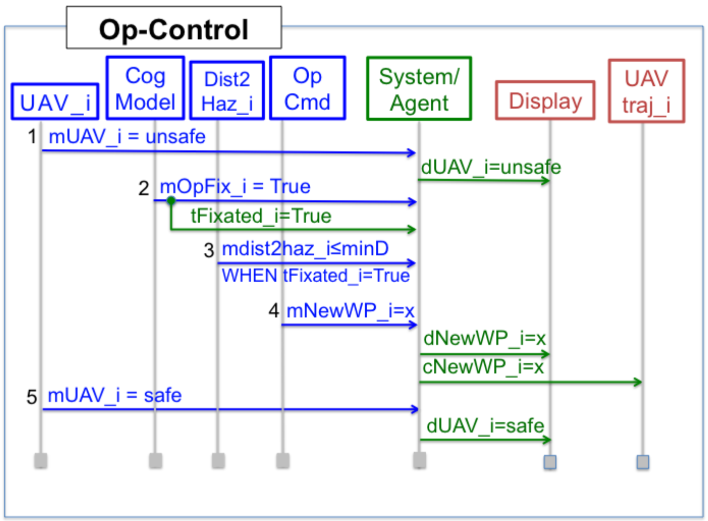
- In software practice, high-quality requirements models are extremely rare
- When they exist, they are often
  - Ambiguous (rep'd in languages which lack a formal semantics)
  - Expressed at a low level of abstraction
- One promising approach: Synthesis of a formal requirements model from **scenarios**
  - Message Sequence Charts (MSCs) –representation of scenarios used by many practitioners to specify software req.
  - Problems with MSCs—no state variables, how to combine them unclear, not formal, no way to express guards, …
  - Some research on synthesizing formal models from MSCs and other representations for scenarios--but unreadable, don't scale…
- Our approach
  - Event Sequence Charts (ESCs), inspired by MSCs
  - A Mode Diagram

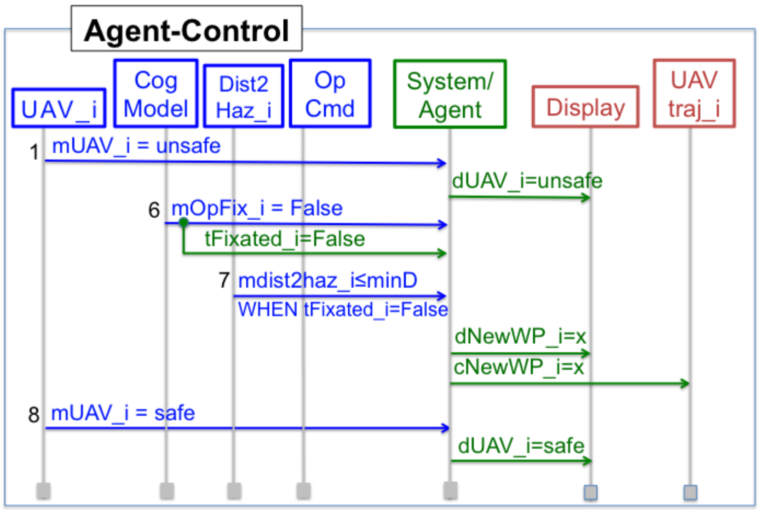
Formal System Model Synthesis: Example of a Scenario Rep'd as an ESC



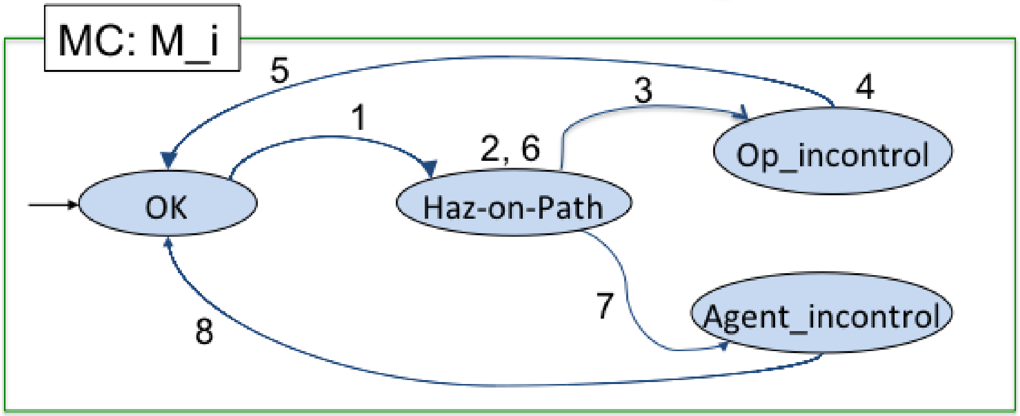
Formal System Model Synthesis: Role of a Mode Diagram

Scenarios specified as ESCs





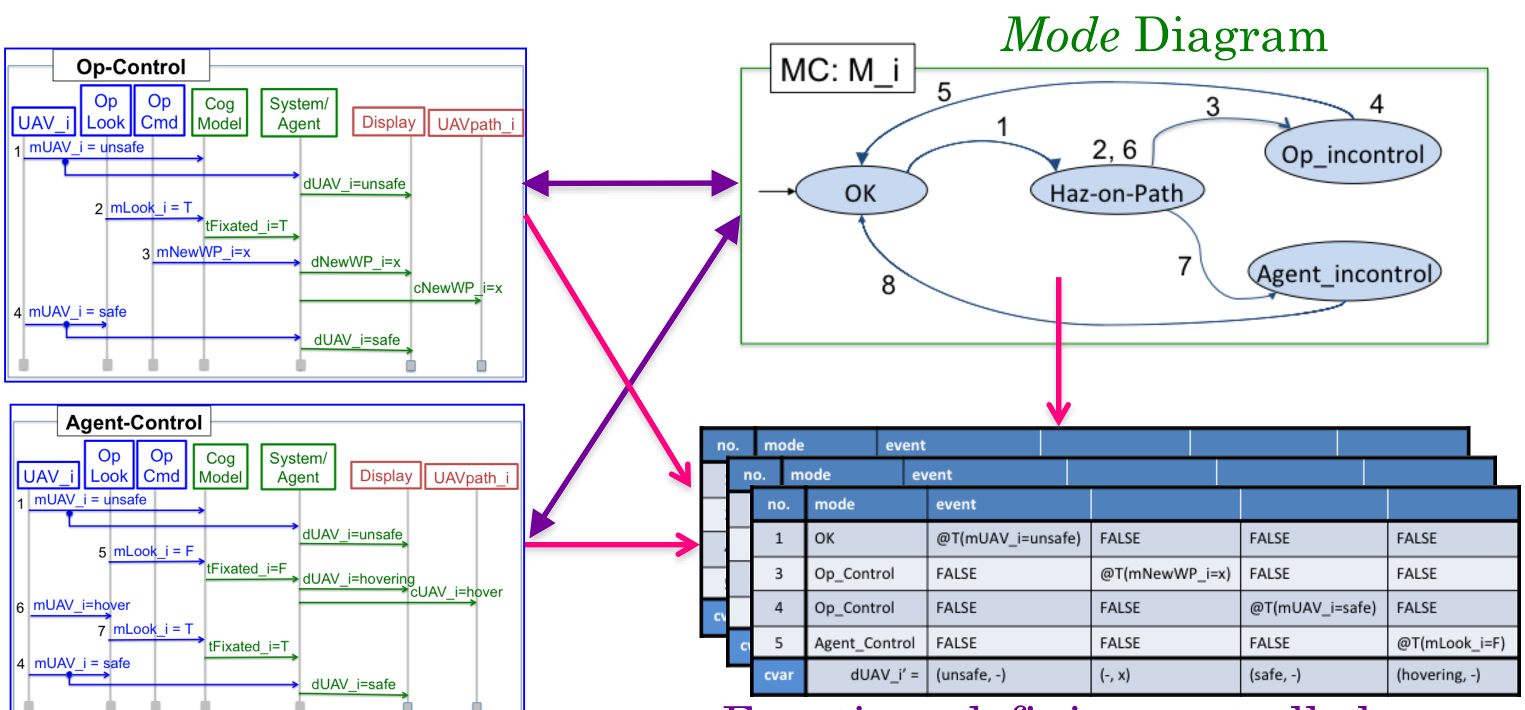
Mode Diagram



Formal System Model Synthesis from Scenarios and a Mode Diagram

- Develop a set of **scenarios** and specify them as ESCs
- Formulate ***mode diagrams*** & relate them to the ESCs
- From the ESCS + mode diagrams, synthesize a **function** for each controlled var
- Synthesize a formal model from the above

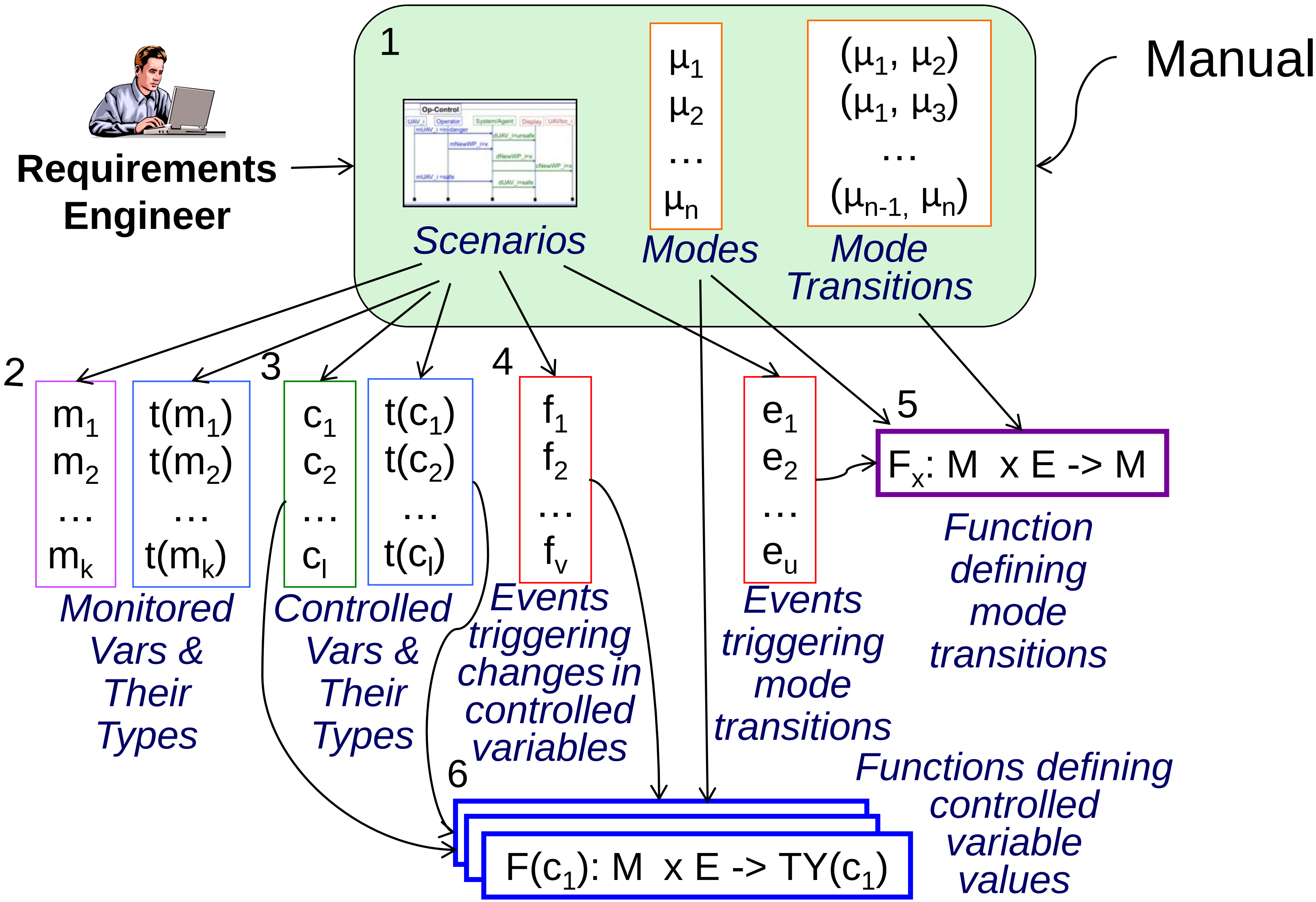
Scenarios specified as ESCs



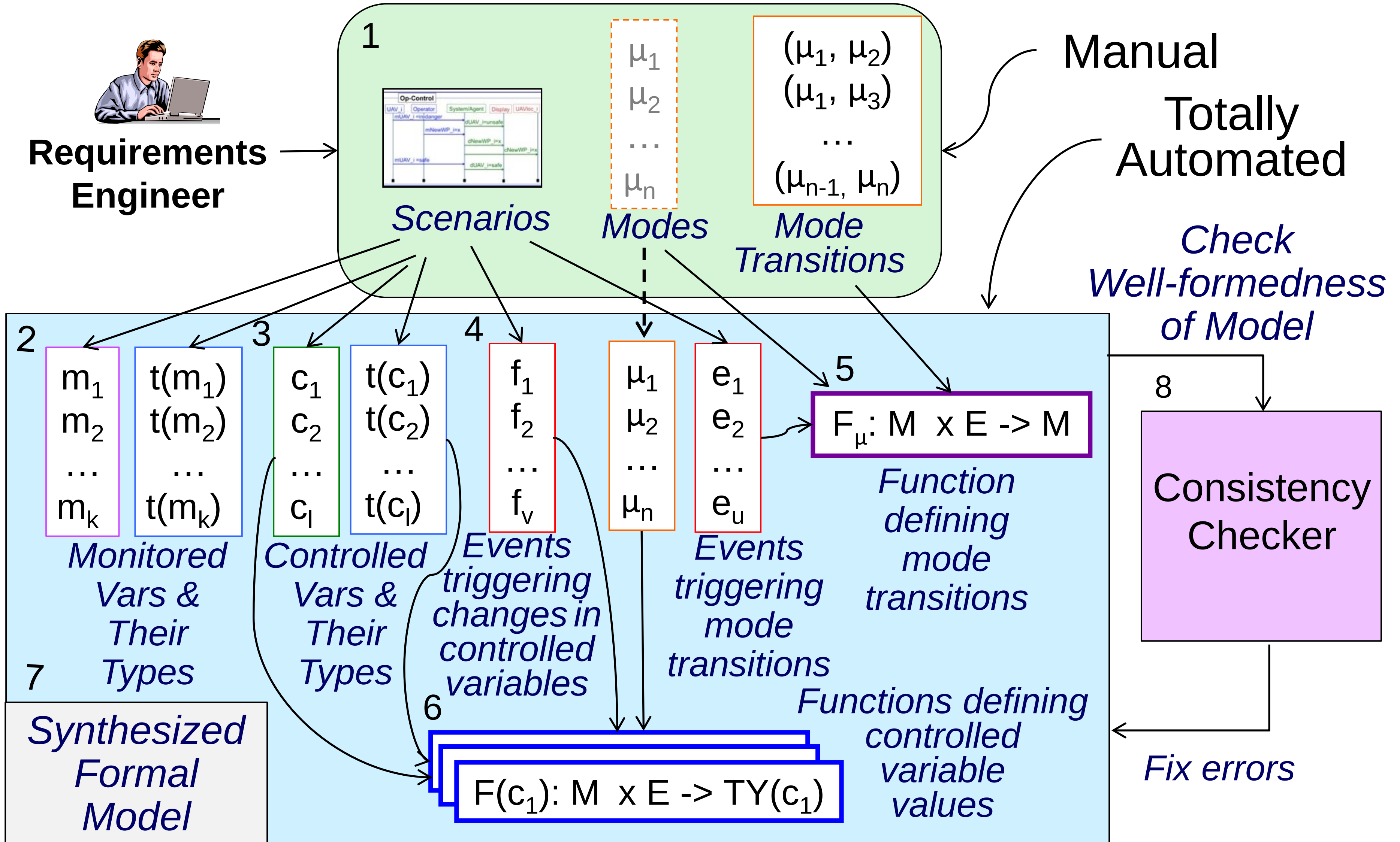
no.	mode	event			
1	OK	@T(mUAV_i=unsafe)	FALSE	FALSE	FALSE
3	Op_Control	FALSE	@T(mNewWP_i=x)	FALSE	FALSE
4	Op_Control	FALSE	FALSE	@T(mUAV_i=safe)	FALSE
5	Agent_Control	FALSE	FALSE	FALSE	@T(mLook_i=F)
cvar	dUAV_i' =	(unsafe, -)	(-, x)	(safe, -)	(hovering, -)

Functions defining controlled vars + other components of a formal model

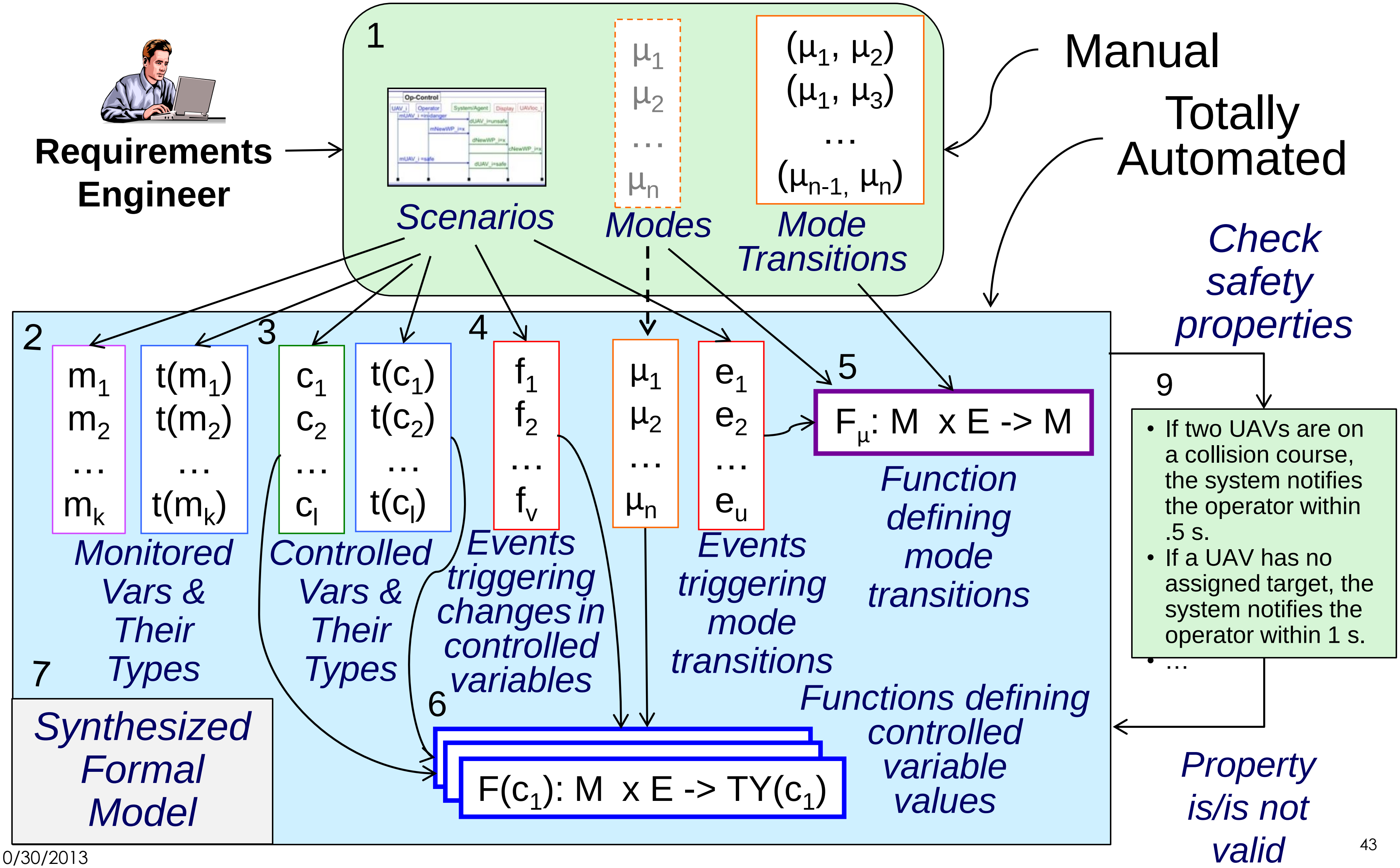
Formal System Model Synthesis: Method



Formal System Model Synthesis: Method



Formal System Model Synthesis: Method



SUMMARY

- The size & complexity of software systems continue to grow
- Formal requirements models have many benefits
  - Should be readable
  - Should be analyzable
  - Should scale to handle large, complex systems
  - If possible, executable
- System-level abstractions, e.g., modes, can help in building formal requirements models for large, complex systems
- Formal requirements models are still rare in industry
- Promising approach: scenario-based synthesis of formal requirements models

A formal system requirements model provides a solid foundation for building a **safe** software system

CYBER PHYSICAL SYSTEMS

WE NEED FORMAL MODELS OF CYBER PHYSICAL SYSTEMS

- Key Challenge: How to integrate physical dynamics, which operate in a temporal and spatial continuum, with software's digital behavior
- Digital software systems can be modeled as a relation on monitored and controlled variables
- Physical quantities in the system environment are often better modeled using a declarative technique based on conservation laws.
  - These techniques are often called "equational," because a connection between components specifies equalities of dynamically varying quantities (Edward Lee)
- Issues
  - How to import digital models of the system req. into physical models
  - How to import physical models into digital models of the system req.

ADDRESSING THE SOFTWARE PROBLEM

- The size and complexity of software system have grown enormously
- Software failures are continuing to occur in many critical applications

PROBLEM

- NEED METHODS FOR MODELING AND SPECIFYING THE REQUIREMENTS OF LARGE, COMPLEX SOFTWARE SYSTEMSAmong the most important models are requirements models--**formal system requirements models**

Premise

A solid foundation for building a
safe software system is
a precise, unambiguous
system requirements model