Compositional Analysis of System Architectures (using Lustre)

Mike Whalen

Program Director
University of Minnesota Software Engineering Center

UNIVERSITY OF MINNESOTA

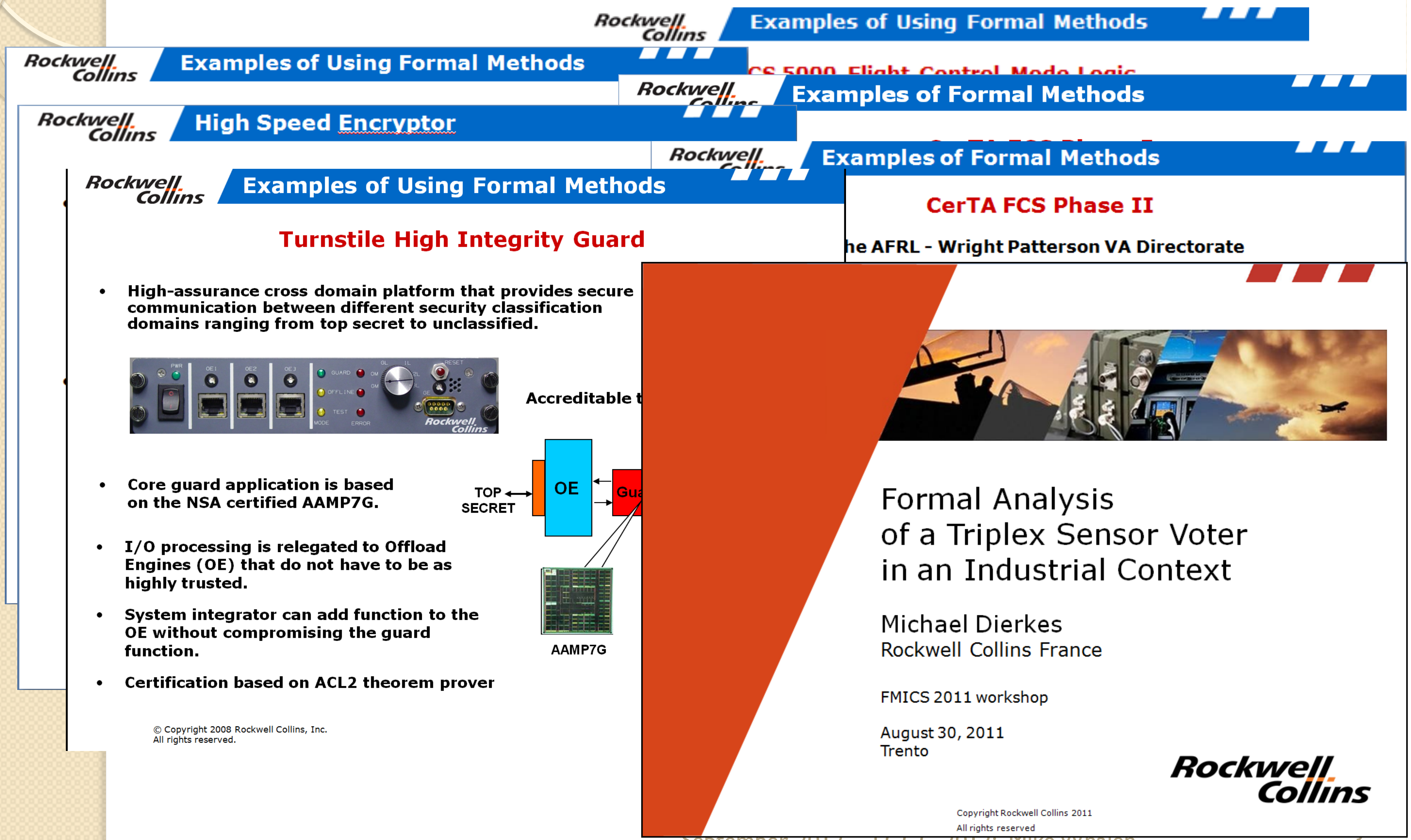
Software Engineering Center

Sponsored by NSF Research Grant CNS-1035715

Acknowledgements

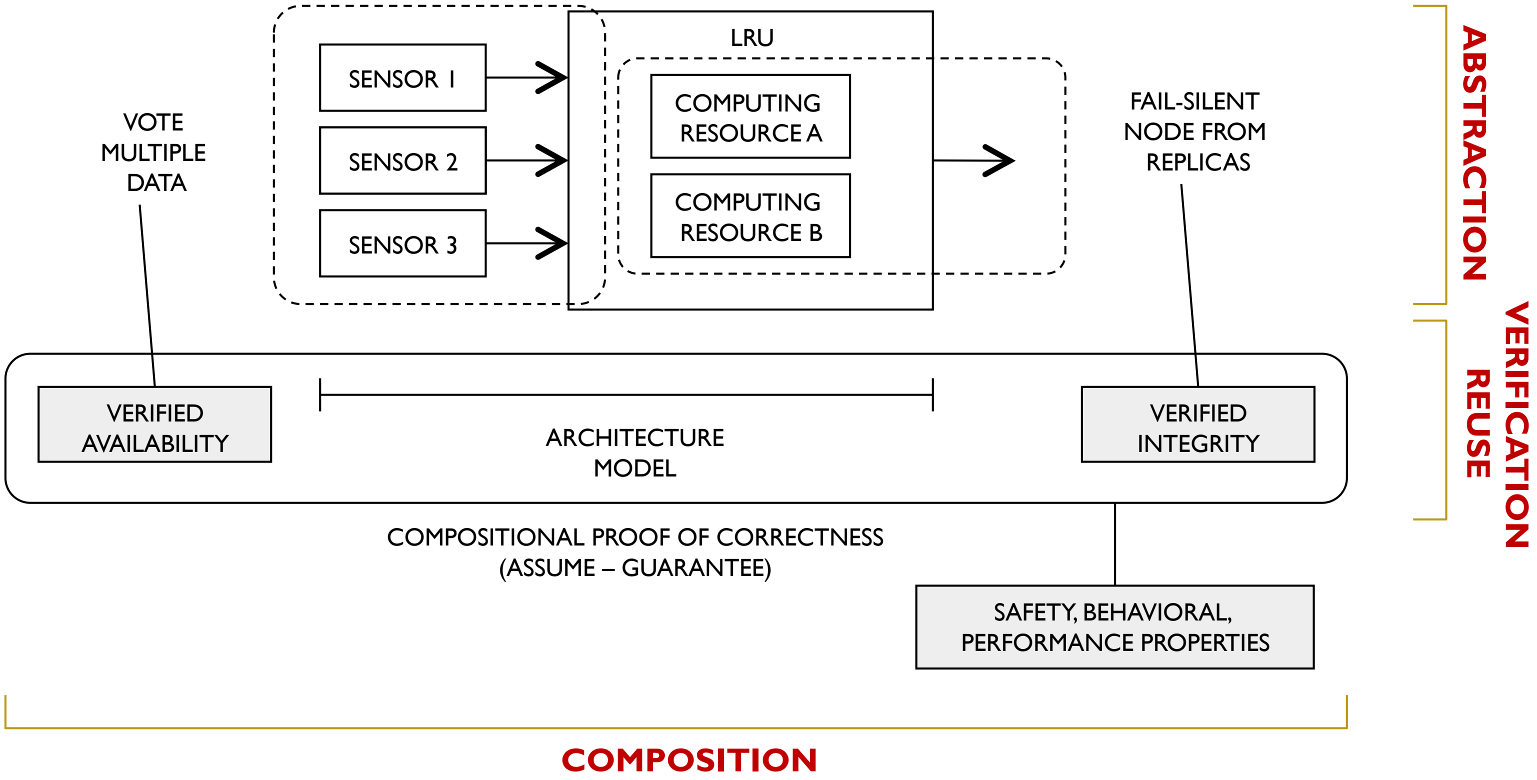
- Rockwell Collins (**Darren Cofer**, **Andrew Gacek**, Steven Miller, Lucas Wagner)
- UPenn: (Insup Lee, Oleg Sokolsky)
- UMN (Mats P. E. Heimdahl)
- CMU SEI (Peter Feiler)

Component Level Formal Analysis Efforts

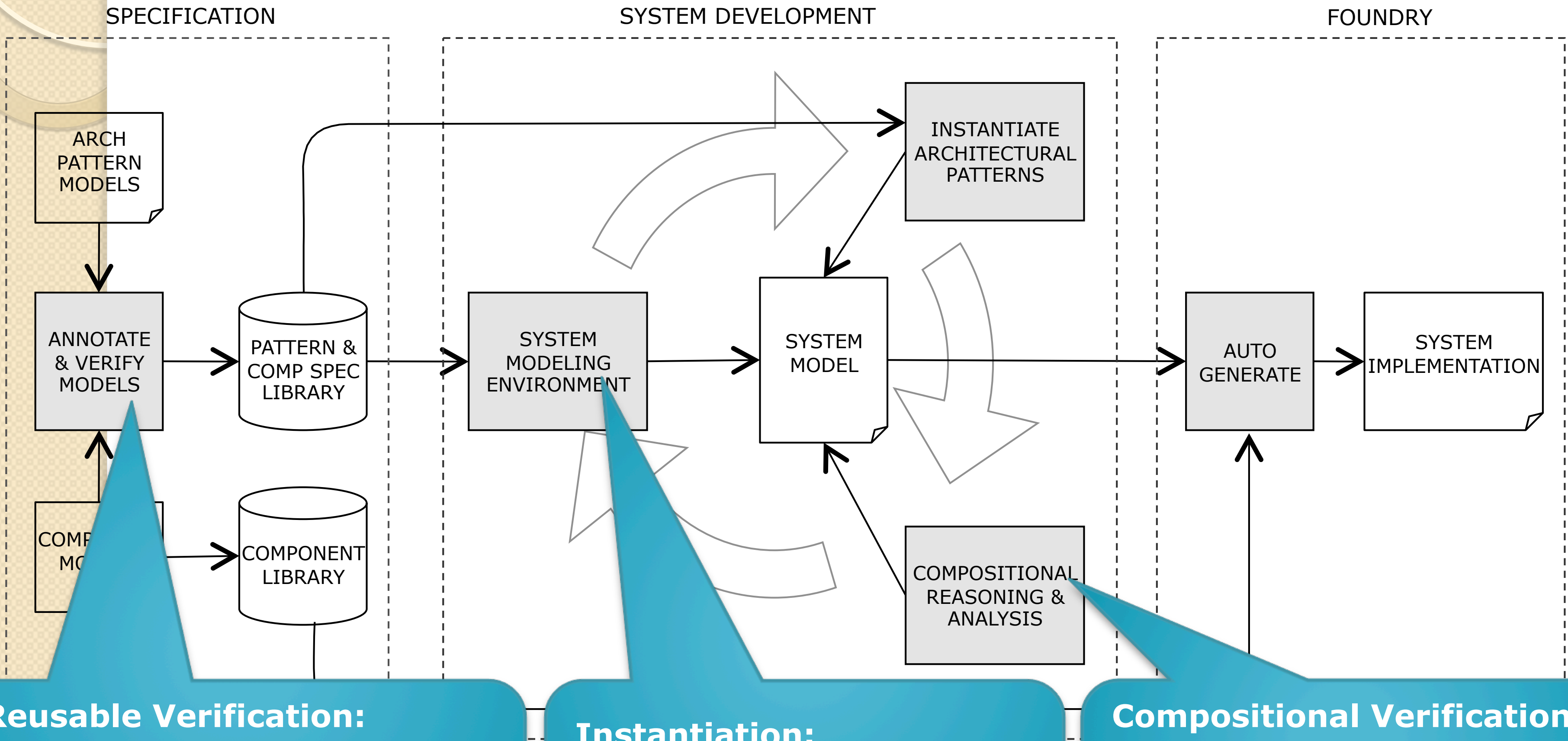


Vision

System design & verification through pattern application and compositional reasoning



System verification



Reusable Verification:
Proof of component and pattern requirements (guarantees) and specification of context (assumptions)

Instantiation:
Check structural constraints, Embed assumptions & guarantees in system model

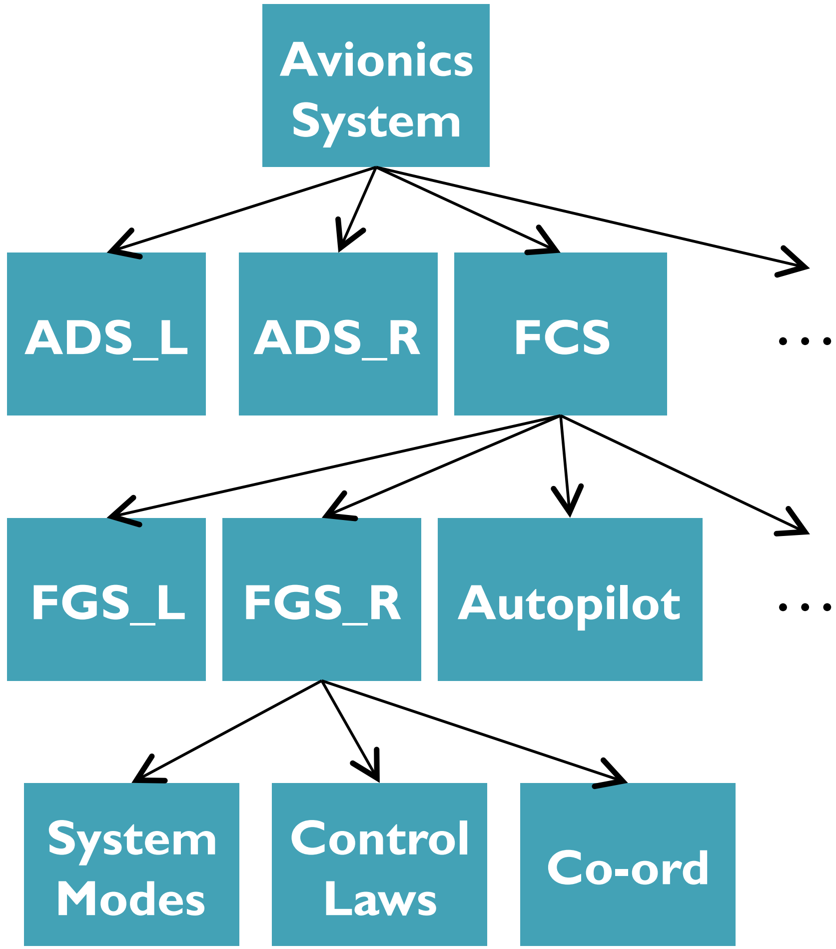
Compositional Verification:
System properties are verified by model checking using component & pattern contracts

Hierarchical reasoning about systems

- Avionics system req

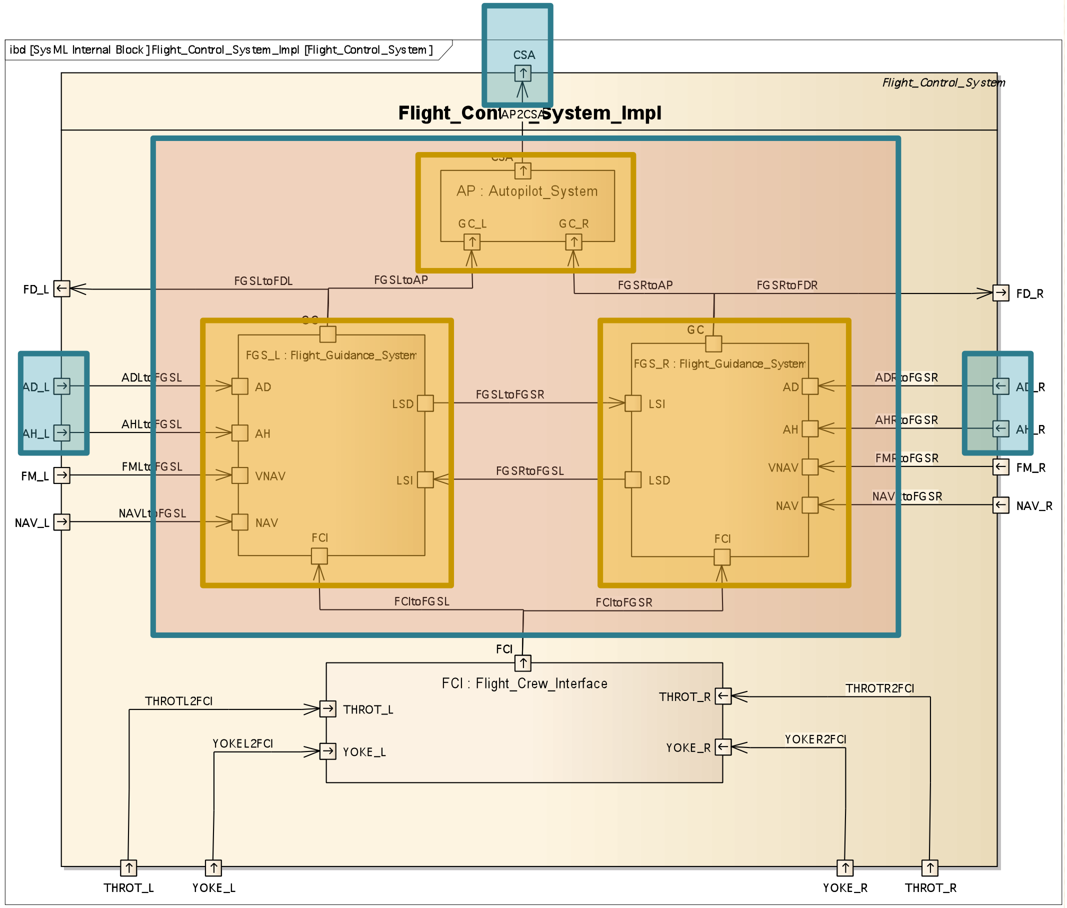
Under single-fault assumption, GC output transient response is bounded in time and magnitude

- Relies upon
  - Accuracy of air data sensors
  - Control commands from FCS
    - Mode of FGS
    - FGS control law behavior
    - Failover behavior between FGS systems
    - ….
  - Response of Actuators
  - Timing/Lag/Latency of Communications



Compositional Reasoning for Active Standby

- Want to prove a **transient response** property
  - The autopilot will not cause a sharp change in pitch of aircraft.
  - Even when one FGS fails and the other assumes control
- Given assumptions about the **environment**
  - The sensed aircraft pitch from the air data system is within some absolute bound and doesn't change too quickly
  - The discrepancy in sensed pitch between left and right side sensors is bounded.
- and guarantees provided by **components**
  - When a FGS is active, it will generate an acceptable pitch rate
- As well as **facts** provided by pattern application
  - Leader selection: at least one FGS will always be active (modulo one "failover" step)




```
transient_response_1 : assert true ->
  abs(CSA.CSA_Pitch_Delta) < CSA_MAX_PITCH_DELTA ;
transient_response_2 : assert true ->
  abs(CSA.CSA_Pitch_Delta - prev(CSA.CSA_Pitch_Delta, 0.0))
     < CSA_MAX_PITCH_DELTA_STEP ;
```

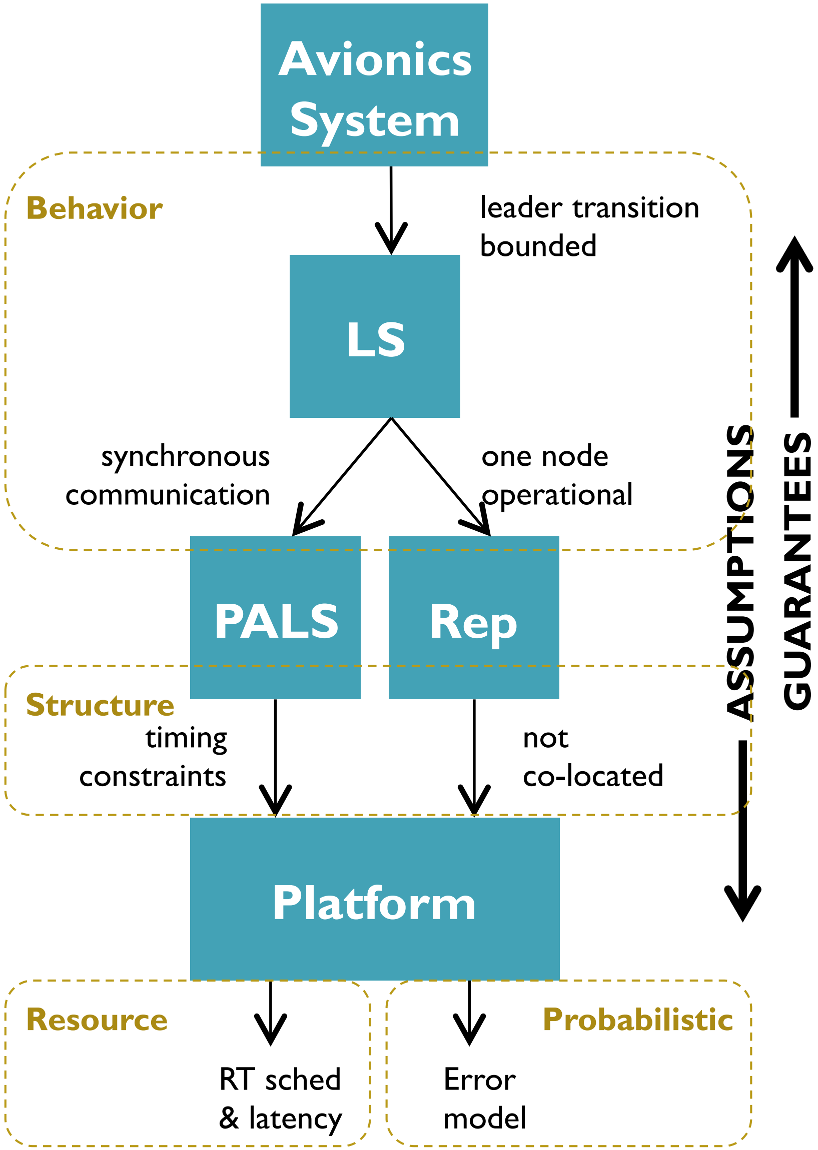
Contracts between patterns and components

- Avionics system requirement

Under single-fault assumption, GC output transient response is bounded in time and magnitude

- Relies upon
  - Guarantees provided by patterns and components
  - Structural properties of model
  - Resource allocation feasibility
  - Probabilistic system-level failure characteristics

Principled mechanism for "passing the buck"



Contracts

- Derived from Lustre and Property Specification Language (PSL) formalism
  - IEEE standard
  - In wide use for hardware verification
- Assume / Guarantee style specification
  - Assumptions: “Under these conditions”
  - Promises (Guarantees):“… the system will do X”
- Local definitions can be created to simplify properties

```
Contract:

fun abs(x: real) : real = if (x > 0) then x else -x ;

const ADS_MAX_PITCH_DELTA: real = 3.0 ;
const FCS_MAX_PITCH_SIDE_DELTA: real = 2.0 ;
…

property AD_L_Pitch_Step_Delta_Valid =
  true ->
    abs(AD_L.pitch.val - prev(AD_L.pitch.val, 0.0)) <
        ADS_MAX_PITCH_DELTA ;

…

active_assumption: assume some_fgs_active ;

transient_assumption :
  assume AD_L_Pitch_Step_Delta_Valid and
         AD_R_Pitch_Step_Delta_Valid and Pitch_lr_ok ;


transient_response_1 :
  assert true -> abs(CSA.CSA_Pitch_Delta) <
                     CSA_MAX_PITCH_DELTA ;
transient_response_2 :
  assert true ->
      abs(CSA.CSA_Pitch_Delta -
          prev(CSA.CSA_Pitch_Delta, 0.0)) <
               CSA_MAX_PITCH_DELTA_STEP ;
```

Reasoning about contracts

- Notionally: *It is always the case that if the component assumption is true, then the component will ensure that the guarantee is true.*
  - $\cancel{G(A \Rightarrow P)}$;
- An assumption violation in the past may prevent component from satisfying current guarantee, so we need to assert that the assumptions are true up to the current step:
  - $G(H(A) \Rightarrow P)$;

Reasoning about Contracts

- Given the set of component contracts:
  $\Gamma = \{ G(H(A_c) \Rightarrow P_c) \mid c \in C \}$
- Architecture adds a set of obligations that tie the system assumption to the component assumptions

$$Q = \{H(A_s) \implies P_s\} \cup \{H(A_s) \implies A_c \mid c \in C\}$$

- This process can be repeated for any number of abstraction levels

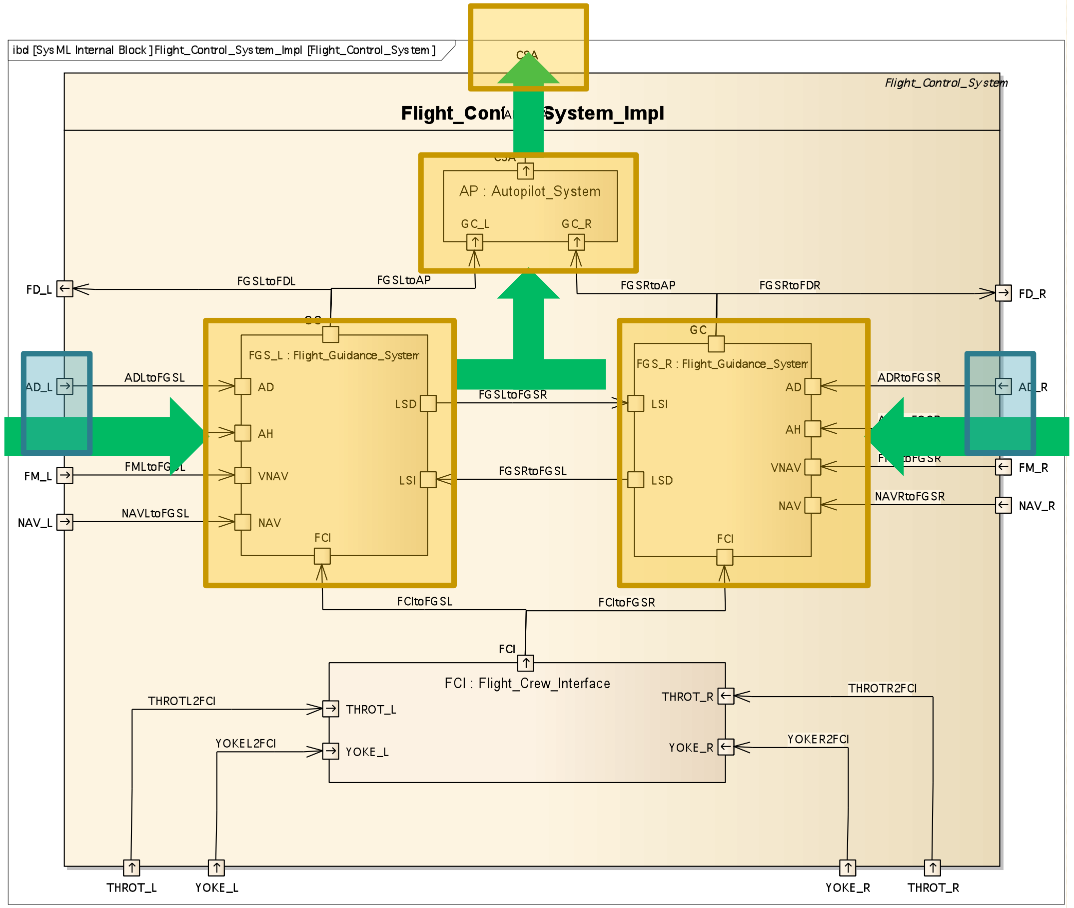
Composition Formulation

- Suppose we have
  - *Sets of formulas* Γ *and* Q
  - *A well-founded order* $\prec$ *on* Q
  - *Sets* $\Theta_q \subseteq \Delta_q \subseteq Q$, *such that* $r \in \Theta_q$ *implies* $r \prec q$
- Then if for all $q \in Q$
  - $\Gamma \Rightarrow G((Z(H(\Theta_q)) \wedge \Delta_q) \Rightarrow q)$
- Then:

  $G(q)$ for all $q \in Q$
- [Adapted from McMillan]

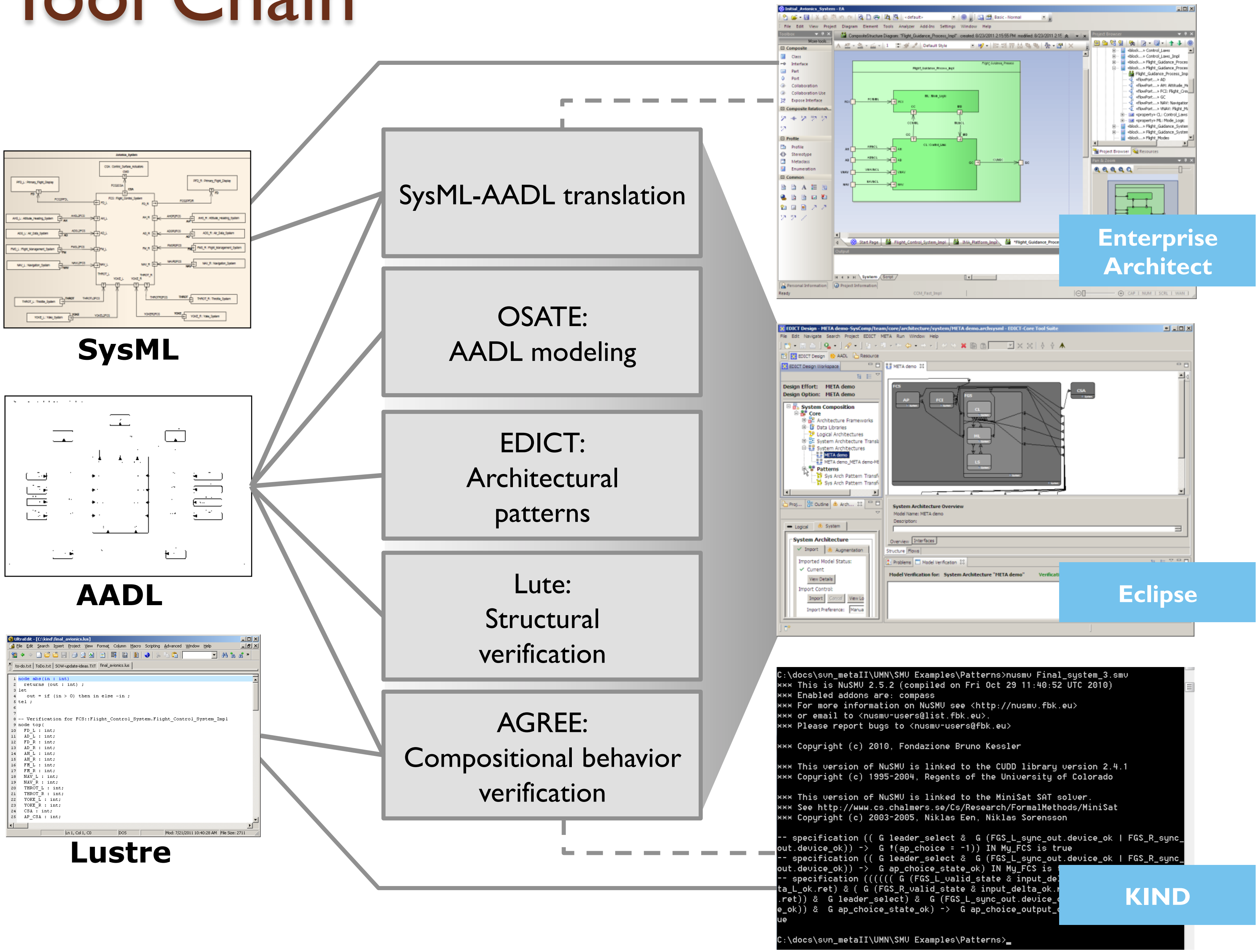
A concrete example

- Order of data flow through system components is computed by reasoning engine
  - {System inputs} → {FGS_L, FGS_R}
  - {FGS_L, FGS_R} → {AP}
  - {AP} → {System outputs}



- Based on flow, we establish four proof obligations
  - System assumptions → FGS_L assumptions
  - System assumptions → FGS_R assumptions
  - System assumptions + FGS_L guarantees + FGS_R guarantees → AP assumptions
  - System assumptions + {FGS_L, FGS_R, AP} guarantees → System guarantees
- System can handle circular flows, but user has to choose where to break cycle

Tool Chain



SysML
AADL
Lustre
SysML-AADL translation
OSATE:
AADL modeling
EDICT:
Architectural
patterns
Lute:
Structural
verification
AGREE:
Compositional behavior
verification
Enterprise
Architect
Eclipse
KIND

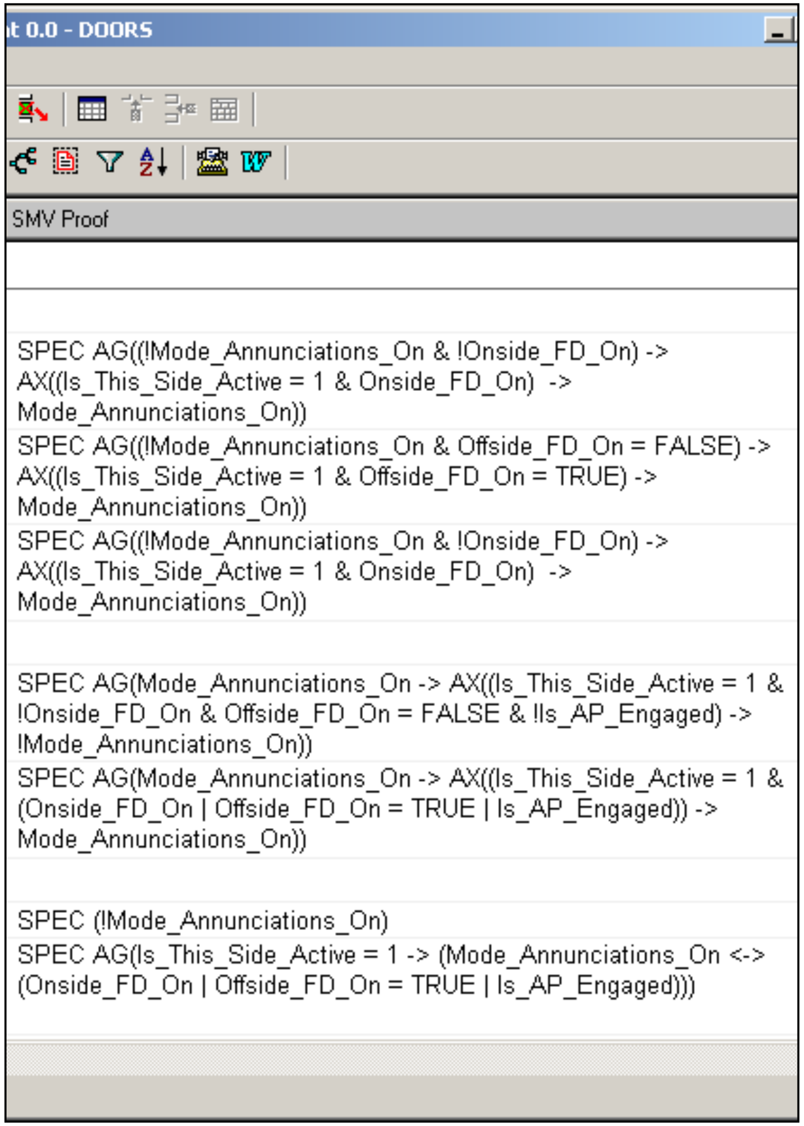
Research Challenges

Proving

- Current back-end analysis performed using SMT-based k-induction model checking technique [Hagen and Tinelli: FMCAD 2008]
- Very scalable if properties can be inductively proven
- Unfortunately, Inductive proofs often fail because properties are too weak
- Lots of work on lemma/invariant discovery to strengthen properties
  - Bjesse and Claessen: *SAT-based verification without State Space Traversal*
  - Bradley: *SAT-based Model Checking without Unrolling*
  - Tinelli: *Instantiation-Based Invariant Discovery* [NFM 2011]
- These strengthening methods are not targeted towards our problem
- Only supports analysis of linear models

Scaling

- What do you do when systems and subcomponents have hundreds of requirements?
  - FGS mode logic: 280 requirements
  - DWM: >600 requirements
- Need to create automated slicing techniques for predicates rather than code.
  - Perhaps this will be in the form of counterexample-guided refinement



Assigning blame

- Counterexamples are often hard to understand for big models
- It is much worse (in my experience) for property-based models
- Given a counterexample, can you automatically assign blame to one or more subcomponents?
- Given a "blamed" component, can you automatically open the black box to strengthen the component guarantee?

Signal	**Step...**					
	0	1	2	3	4	5
AD_L.pitch.val	-0.91	-1.83	-2.74	-3.65	-4.35	-4.39
AD_L.pitch.valid	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
AD_R.pitch.val	0.83	-0.09	-1.00	-1.91	-2.83	-3.74
AD_R.pitch.valid	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
AP.CSA.csa_pitch_delta	0.00	0.13	0.09	0.26	0.74	-4.26
AP.GC_L.cmds.pitch_delta	0.00	-4.91	-4.65	-4.57	-4.74	-4.35
AP.GC_L.mds.active	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
AP.GC_R.cmds.pitch_delta	0.00	0.83	-4.43	-4.48	4.91	4.83
AP.GC_R.mds.active	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
Assumptions for AP	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FCI	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FGS_L	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Assumptions for FGS_R	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
FGS_L.GC.cmds.pitch_delta	-4.91	-4.65	-4.57	-4.74	-4.35	0.09
FGS_L.GC.mds.active	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
FGS_L.LSO.leader	2	2	3	2	1	3
FGS_L.LSO.valid	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FGS_R.GC.cmds.pitch_delta	0.83	-4.43	-4.48	4.91	4.83	3.91
FGS_R.GC.mds.active	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
FGS_R.LSO.leader	0	0	1	0	1	1
FGS_R.LSO.valid	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
leader_pitch_delta	0.00	0.83	0.83	0.83	0.83	-4.35
System level guarantees	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE

"Argument Engineering"

- Disparate kinds of evidence throughout the system
  - Probabilistic
  - Resource
  - Structural properties of model
  - Behavioral properties of model
- How do we tie these things together?
- Evidence graph, similar to proof graph in PVS
  - Shows evidential obligations that have not been discharged

Dealing with Time

- Current analysis is synchronous
  - It assumes all subcomponents run at the same rate
  - It assumes single-step delay between subcomponents
- This is not how the world works!
  - …unless you use Time-Triggered Architectures or PALS
- Adding more realistic support for time is crucial to accurate analyses
  - Time intervals tend to diverge in hierarchical verification
  - E.g. synchronization.

Provocations

We do not yet have a clear idea of how to effectively partition system analyses to perform effective compositional reasoning across domains

We need research to combine analyses to make overall system analysis more effective.

The Collins/UMN META tools are a first step towards this goal.

Conclusions

- Still early work…
  - Many AADL constructs left to be mapped
  - Many timing issues need to be resolved
  - Better support for proof engineering needs to be found
- **But**
  - Already can do some interesting analysis with tools
  - Sits in a nice intersection between requirements engineering and formal methods
  - Lots of work yet on how best to specify requirements

Thank you!

Ευχαριστώ
Merci
Grazie
Díky
Gracias
Vielen Dank
Obrigado!
Teşekkürler
شكراً
धन्यवाद