



# Compositional Verification of Architectural Models

High Confidence Software & Systems Conference

9 May 2012  
Darren Cofer



**Rockwell  
Collins**

## Outline

- What is META?
- Project vision
- Tool environment
- Technologies
  - System-level modeling and translation
  - Complexity-reducing architectural patterns
  - Compositional verification
- Next steps

## Team

- Rockwell Collins / Advanced Technology Center
  - Darren Cofer, Steven Miller, Andrew Gacek
  - System modeling & analysis, tooling, integration
- UIUC
  - Lui Sha
  - Design pattern development
- University of MN
  - Michael Whalen
  - Pattern verification, compositional analysis
- WWTG
  - Chris Walter, Brian LaValley
  - Pattern implementation & analysis tools



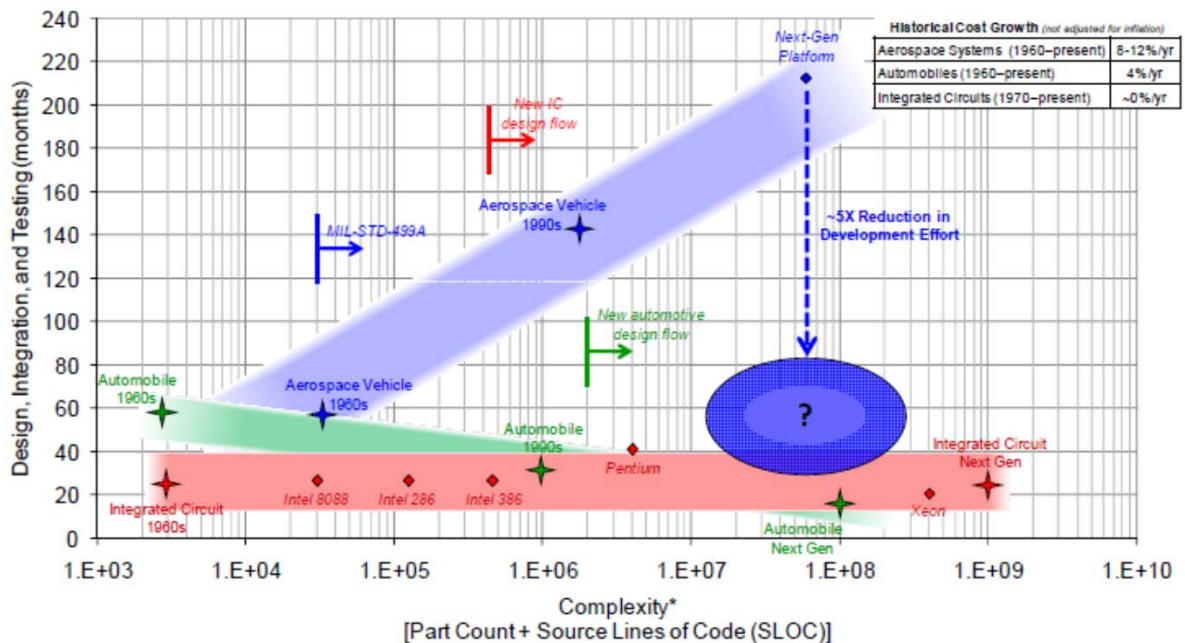
## What is META?

- Devise, implement, and demonstrate a radically different approach to the **design**, **integration**/manufacturing, and **verification** of defense systems/vehicles
- Enhance designer's ability to manage system complexity
- "Foundry-style" model of manufacturing
- Five technical areas
  1. Metrics of complexity
  2. Metrics of adaptability
  3. Meta-language for system design
  4. Design flow & tools
  5. Verification flow & tools



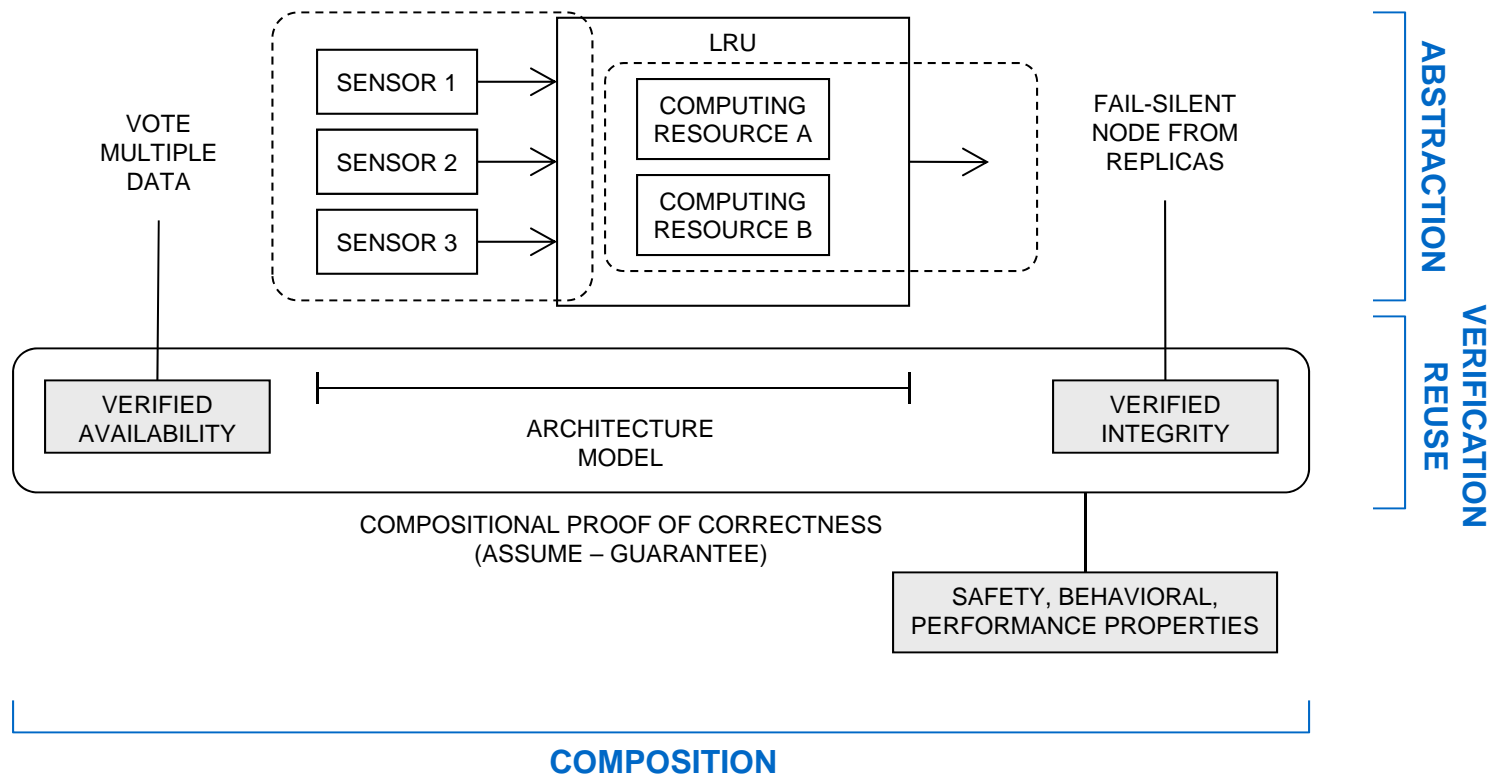
APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

### Historical schedule trends with complexity

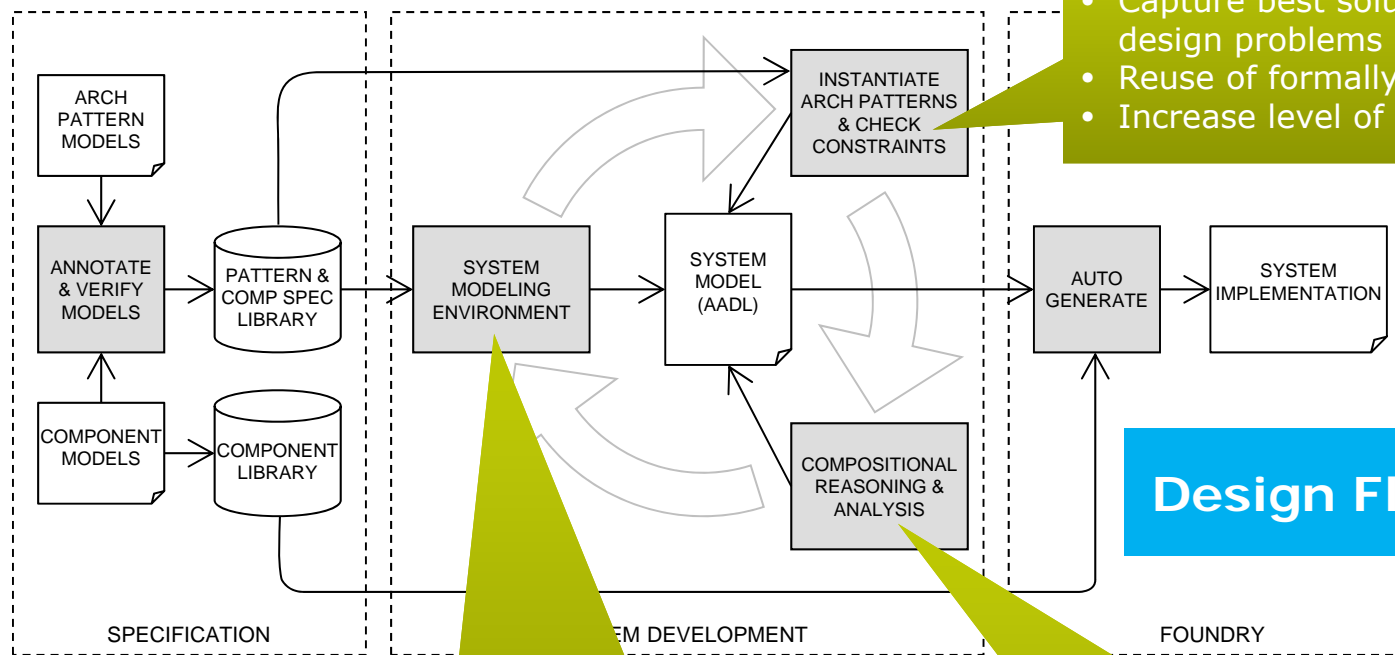


## Vision

- Improve effectiveness and scalability of system design and verification through pre-verified design patterns and compositional reasoning



## Approach



### Complexity-reducing design patterns

- Capture best solutions to architectural design problems
- Reuse of formally verified solutions
- Increase level of design abstraction **2**

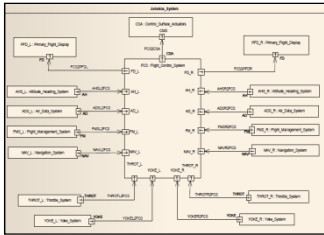
### System architecture modeling

- Apply formal specification and analysis tools to system-level design
- Separate component specification and implementation
- Automated model translation **1**

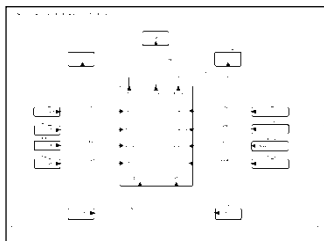
### Compositional verification

- Reason about system behavior based on contracts and system design model structure
- Compositional approach scales to large software systems **3**

# Tool chain



**SysML**



**AADL**

```

1 #include "lstm/asm/asm.h"
2 #include "lstm/asm/asm.h"
3 #include "lstm/asm/asm.h"
4 #include "lstm/asm/asm.h"
5 #include "lstm/asm/asm.h"
6 #include "lstm/asm/asm.h"
7 #include "lstm/asm/asm.h"
8 #include "lstm/asm/asm.h"
9 #include "lstm/asm/asm.h"
10 #include "lstm/asm/asm.h"
11 #include "lstm/asm/asm.h"
12 #include "lstm/asm/asm.h"
13 #include "lstm/asm/asm.h"
14 #include "lstm/asm/asm.h"
15 #include "lstm/asm/asm.h"
16 #include "lstm/asm/asm.h"
17 #include "lstm/asm/asm.h"
18 #include "lstm/asm/asm.h"
19 #include "lstm/asm/asm.h"
20 #include "lstm/asm/asm.h"
21 #include "lstm/asm/asm.h"
22 #include "lstm/asm/asm.h"
23 #include "lstm/asm/asm.h"
24 #include "lstm/asm/asm.h"
25 #include "lstm/asm/asm.h"
26 #include "lstm/asm/asm.h"
27 #include "lstm/asm/asm.h"
28 #include "lstm/asm/asm.h"
29 #include "lstm/asm/asm.h"
30 #include "lstm/asm/asm.h"
31 #include "lstm/asm/asm.h"
32 #include "lstm/asm/asm.h"
33 #include "lstm/asm/asm.h"
34 #include "lstm/asm/asm.h"
35 #include "lstm/asm/asm.h"
36 #include "lstm/asm/asm.h"
37 #include "lstm/asm/asm.h"
38 #include "lstm/asm/asm.h"
39 #include "lstm/asm/asm.h"
40 #include "lstm/asm/asm.h"
41 #include "lstm/asm/asm.h"
42 #include "lstm/asm/asm.h"
43 #include "lstm/asm/asm.h"
44 #include "lstm/asm/asm.h"
45 #include "lstm/asm/asm.h"
46 #include "lstm/asm/asm.h"
47 #include "lstm/asm/asm.h"
48 #include "lstm/asm/asm.h"
49 #include "lstm/asm/asm.h"
50 #include "lstm/asm/asm.h"
51 #include "lstm/asm/asm.h"
52 #include "lstm/asm/asm.h"
53 #include "lstm/asm/asm.h"
54 #include "lstm/asm/asm.h"
55 #include "lstm/asm/asm.h"
56 #include "lstm/asm/asm.h"
57 #include "lstm/asm/asm.h"
58 #include "lstm/asm/asm.h"
59 #include "lstm/asm/asm.h"
60 #include "lstm/asm/asm.h"
61 #include "lstm/asm/asm.h"
62 #include "lstm/asm/asm.h"
63 #include "lstm/asm/asm.h"
64 #include "lstm/asm/asm.h"
65 #include "lstm/asm/asm.h"
66 #include "lstm/asm/asm.h"
67 #include "lstm/asm/asm.h"
68 #include "lstm/asm/asm.h"
69 #include "lstm/asm/asm.h"
70 #include "lstm/asm/asm.h"
71 #include "lstm/asm/asm.h"
72 #include "lstm/asm/asm.h"
73 #include "lstm/asm/asm.h"
74 #include "lstm/asm/asm.h"
75 #include "lstm/asm/asm.h"
76 #include "lstm/asm/asm.h"
77 #include "lstm/asm/asm.h"
78 #include "lstm/asm/asm.h"
79 #include "lstm/asm/asm.h"
80 #include "lstm/asm/asm.h"
81 #include "lstm/asm/asm.h"
82 #include "lstm/asm/asm.h"
83 #include "lstm/asm/asm.h"
84 #include "lstm/asm/asm.h"
85 #include "lstm/asm/asm.h"
86 #include "lstm/asm/asm.h"
87 #include "lstm/asm/asm.h"
88 #include "lstm/asm/asm.h"
89 #include "lstm/asm/asm.h"
90 #include "lstm/asm/asm.h"
91 #include "lstm/asm/asm.h"
92 #include "lstm/asm/asm.h"
93 #include "lstm/asm/asm.h"
94 #include "lstm/asm/asm.h"
95 #include "lstm/asm/asm.h"
96 #include "lstm/asm/asm.h"
97 #include "lstm/asm/asm.h"
98 #include "lstm/asm/asm.h"
99 #include "lstm/asm/asm.h"
100 #include "lstm/asm/asm.h"

```

**Lustre**

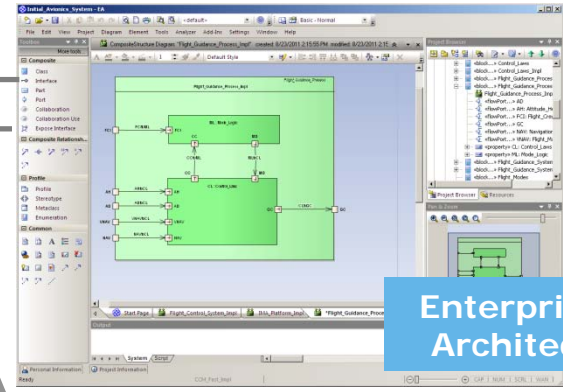
**SysML-AADL translation**

**OSATE:  
AADL modeling**

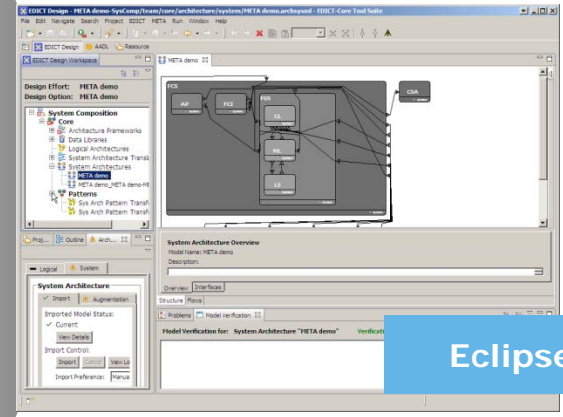
**EDICT:  
Architectural patterns**

**Lute:  
Structural verification**

**AGREE:  
Compositional behavior verification**



**Enterprise Architect**



**Eclipse**

```

C:\docs\svn_metaII\UMM\SMU_Examples\Patterns\nusmu_Final_system_3.osu
*** This is NuSMV 2.5.2 (compiled on Fri Oct 29 11:40:52 UTC 2010)
*** Enabled addons are: cospass
*** For more information on NuSMV see (http://nusmv.fbk.eu)
*** or email to <nusmv-users@fbk.eu>
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

-- specification (( G leader_select & G (FGS_L_sync_out.device_ok | FGS_R_sync_out.device_ok) ) -> G (f{ap_choice = -1} IN My_FCS is true
-- specification (( G leader_select & G (FGS_L_sync_out.device_ok | FGS_R_sync_out.device_ok) ) -> G ap_choice_state_ok IN My_FCS is
-- specification (((((( G (FGS_L_valid_state & input_device_ok) & ( G (FGS_R_valid_state & input_device_ok) & G leader_select) & G (FGS_L_sync_out.device_e_ok) & G ap_choice_state_ok) -> G ap_choice_output_ue

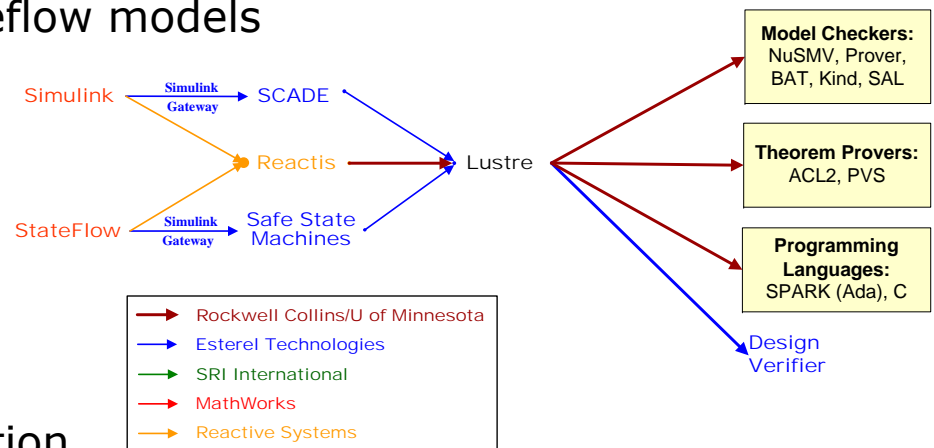
C:\docs\svn_metaII\UMM\SMU_Examples\Patterns>

```

**KIND**

## System architecture modeling

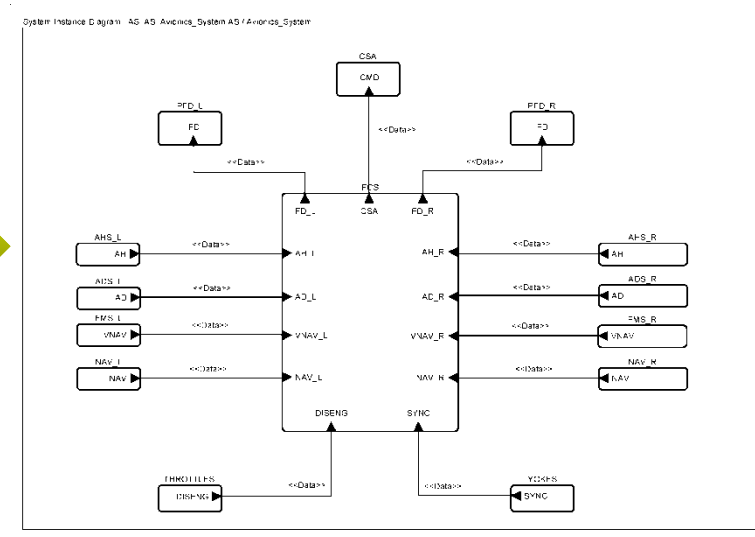
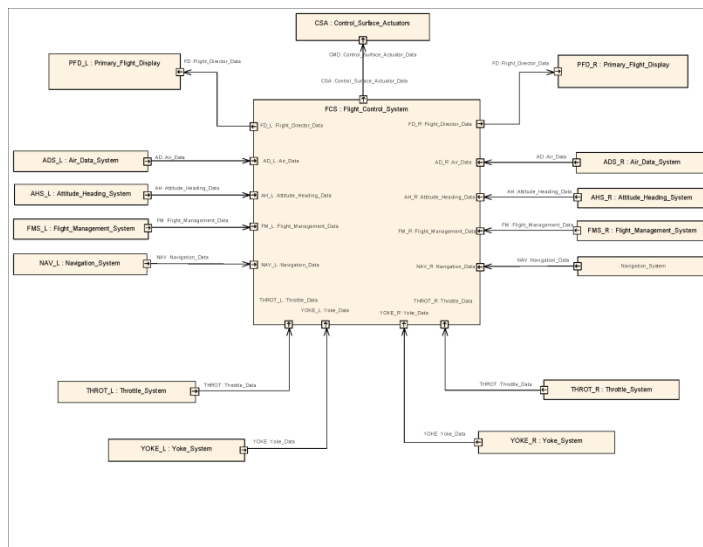
- We have been very successful at applying formal methods to software components produced in model-based development environments
  - Gryphon translation framework
  - Verification of Simulink/Stateflow models
- Objective
  - Leverage this knowledge and apply formal methods to the system design process
- Issues
  - Modeling language and tools
  - Different models of computation
  - Scalability





# System modeling and translation

- AADL is a good fit and provides sufficiently formal notation
  - Available tools do not provide stable graphical environment
  - OSATE: open source, Eclipse-based
- SysML is being adopted by many organizations for system design
  - But has no formal semantics
  - No common textual representation across tools
- Solution: Eclipse plugin that provides bidirectional translation
  - Based on Enterprise Architect SysML tool used by Rockwell Collins
  - Define block stereotypes that correspond to AADL objects



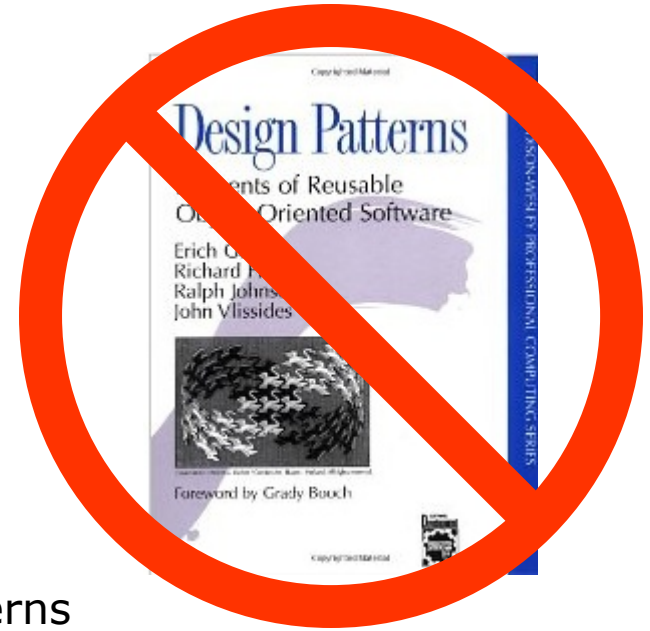
## Scale and composition

- Architectural model does not capture implementation details
  - Component descriptions, interfaces, interconnections
- Assume/guarantee contracts provide the information needed from other modeling domains to reason about system-level properties
  - Guarantees correspond to the component requirements
  - Assumptions correspond to the environmental constraints that were used in proving the component requirements
  - Contract specifies precisely the information that is needed to reason about the component's interaction with other parts of the system
  - Supports hierarchical decomposition of verification process
- Contract can be applied to both components and design patterns
  - Mechanism for *verification reuse*
  - More about this later

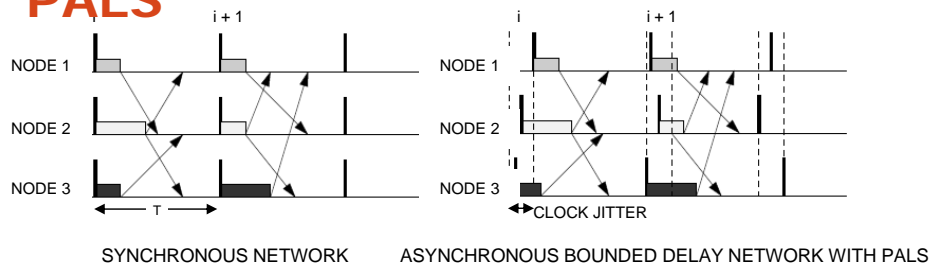
## Complexity-Reducing Architectural Design Patterns

2

- Design pattern = model transformation
  - $p : \mathcal{M} \rightarrow \mathcal{M}$  (partial function)
  - Applied to system models
- Reuse of verification is key
  - Not software reuse
  - Guaranteed behaviors associated with patterns (and components)
- Reduce/manage system complexity
  - Separation of concerns
  - System logic vs. application logic (e.g., fault tolerance)
  - Process complexity vs. design complexity
- Encapsulate & standardize good solutions
  - Raise level of abstraction
  - Codify best practices

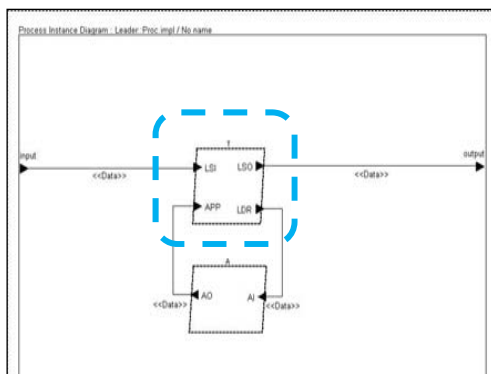


## PALS



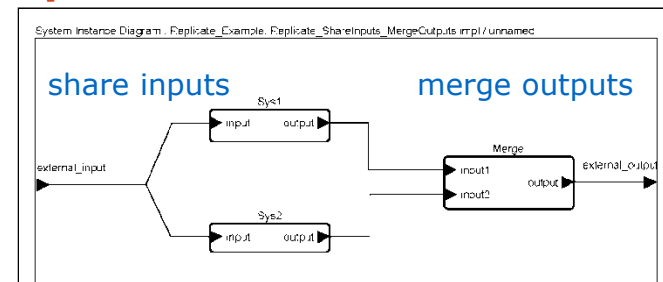
- Provide virtual synchrony for parts of async system
- Assumptions
  - Structural preconditions on system model (bounded jitter, computation, message delivery...)
  - Required data connections exist
- Guarantees
  - Sync logic executes with period T
  - Data from step  $i$  consumed in step  $i + 1$

## Leader Selection



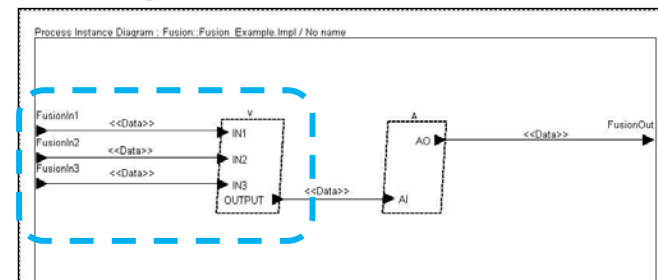
- Create leader for group of nodes
- Assumptions
  - Nodes communicate synchronously
  - At least one non-failed node
- Guarantees
  - All non-failed nodes agree on leader
  - If leader fails, new leader in next step
  - Non-failed node remains leader

## Replication



- Create identical copies of portions of the system
- Assumptions
  - Replicas hosted on platform HW with independent failure modes
- Guarantees
  - One or replicas will operate normally in the event of a single fault

## Voting/Fusion



- Combine several component interfaces
- Assumptions
  - Interfaces terminate at same destination component
  - Interfaces have same data type
- Guarantees
  - Varies with component type
  - Agreement, mid-value select, output select, average

# Initial Avionics System

The screenshot displays the EDICT Design software interface for a system architecture transformation. The main window is titled "System Architecture - Pattern Transform" and shows a complex block diagram of an avionics system. The diagram includes various functional blocks such as CSA, YOKE, THROT, APP, PLS, PCH, FGS, FMS, NAV, PFD, YOKES, THROTLES, and PROC. The blocks are interconnected with lines representing data or control signals.

On the left side, there is a "Design Effort" and "Design Option" section, both set to "META Design Effort" and "META Design Option". Below this is a "System Composition" tree view showing the hierarchy of the system architecture, including "Core", "Data Libraries", "System Architecture Transforms", "Initial Avionics System", and "Patterns".

In the center, a "Pattern Instantiations" table lists 16 instances of patterns used in the system:

C...	S...	S...	Type	Name
1			Replication	Replicate FGS
2			Leader Select	Insert FGS Leader Selection
3			PALS	Apply PALS to FGS Leader Selection
4			Voter Insertion	Insert Guidance Command Selector
5			Replication	Replicate Pitch Sensors
6			Voter Insertion	Insert Pitch Voter
7			Replication	Replicate Airspeed Sensors
8			Voter Insertion	Insert Airspeed Voter
9			Replication	Replicate ADS
10			Replication	Replicate AHS
11			Replication	Replicate FMS
12			Replication	Replicate NAV
13			Replication	Replicate PFD
14			Replication	Replicate Yokes
15			Replication	Replicate Throttles
16			Replication	Replicate Processor

Below the table are "Add", "Edit", and "Remove" buttons. The "Transform Control" section includes "Apply", "Reverse", and "Reset" buttons, along with an "Apply All" button. At the bottom, there are tabs for "Problems", "Model Verification", and "Error Log". The "Model Verification" tab is active, showing the message: "Model verification results are not available."

# Final Avionics System (after pattern transformations)

**System Architecture - Pattern Transform**

Pattern Instantiations:

C...	S...	S...	Type	Name
1	✓		Replication	Replicate FGS
2	✓		Leader Select	Insert FGS Leader Selection
3	✓		PALS	Apply PALS to FGS Leader Selection
4	✓		Voter Insertion	Insert Guidance Command Selector
5	✓		Replication	Replicate Pitch Sensors
6	✓		Voter Insertion	Insert Pitch Voter
7	✓		Replication	Replicate Airspeed Sensors
8	✓		Voter Insertion	Insert Airspeed Voter
9	✓		Replication	Replicate ADS
10	✓		Replication	Replicate AHS
11	✓		Replication	Replicate FMS
12	✓		Replication	Replicate NAV
13	✓		Replication	Replicate PFD
14	✓		Replication	Replicate Yokes
15	✓		Replication	Replicate Throttles
16	✓		Replication	Replicate Processor

System Architecture: Initial\_Avionics\_System  
All PIs Applied

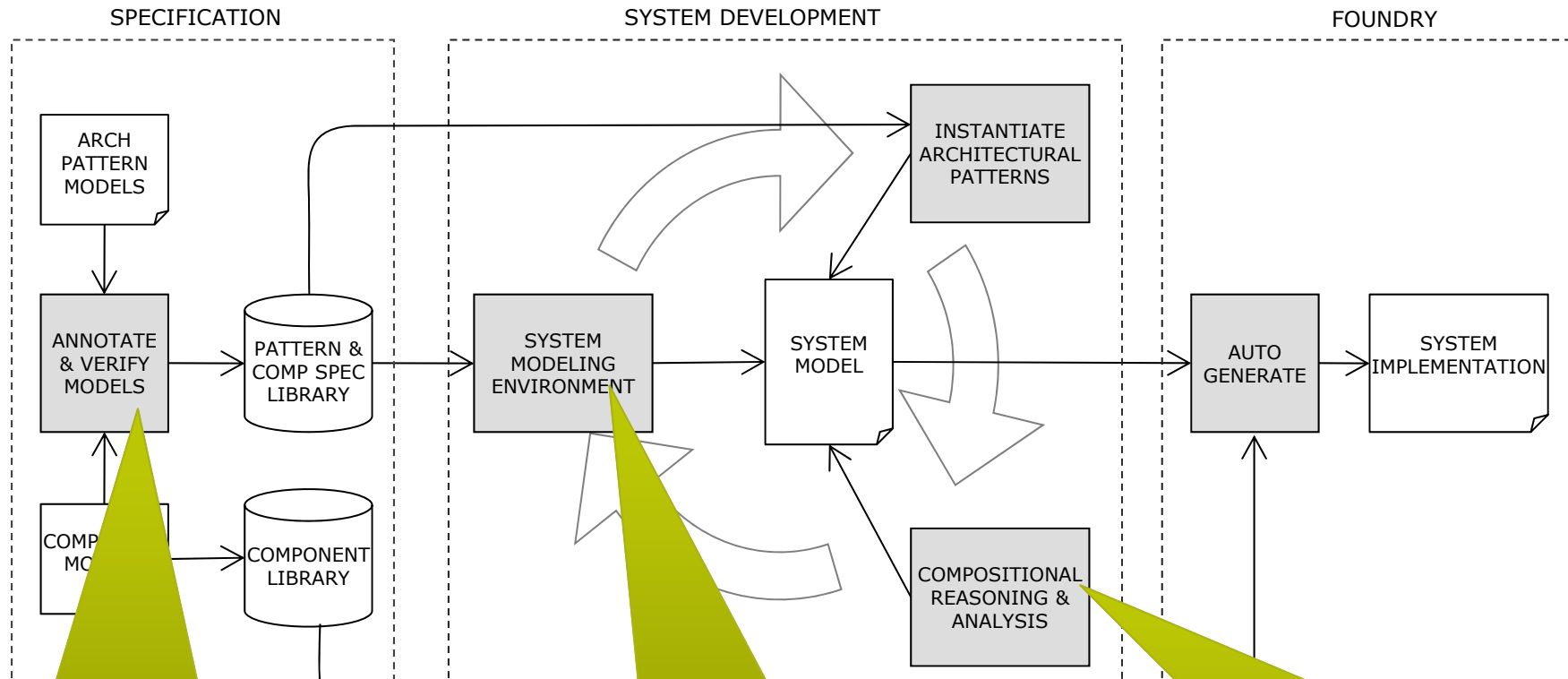
Transform Control: Apply Reverse Reset

Apply All

Model verification results are not available.

# System verification

3



**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

## Categories of system properties

- Structural/static
  - Properties of the transformed model
  - Pattern assumptions, post-conditions
  - Specified and checked using Lute
  - *PALS period constraint*  

$$\text{Deadline} < \text{PALS\_Period} - \text{Max\_Latency} - 2 * \text{Clock\_Jitter}$$
- Behavioral/dynamic
  - Pattern and component interactions
  - Specified in PSL, verified by AGREE using model checking
  - *Failed node will not be leader in next step*  

$$G(!\text{device\_ok}[j] \rightarrow X(\text{leader}[i] \neq j)) ;$$
- Resource allocation
  - RT schedulability, memory allocation, bandwidth allocation
  - ASIIST tool (UIUC/RC)
  - *Threads can be scheduled to meet their deadlines*
- Probabilistic
  - Failure analysis of system
  - Behavior and failure rates described using AADL error annex
  - PRISM/PRISMATIC (SIFT/RC)
  - *$P(\text{all sensors failed}) < 10^{-9}$*



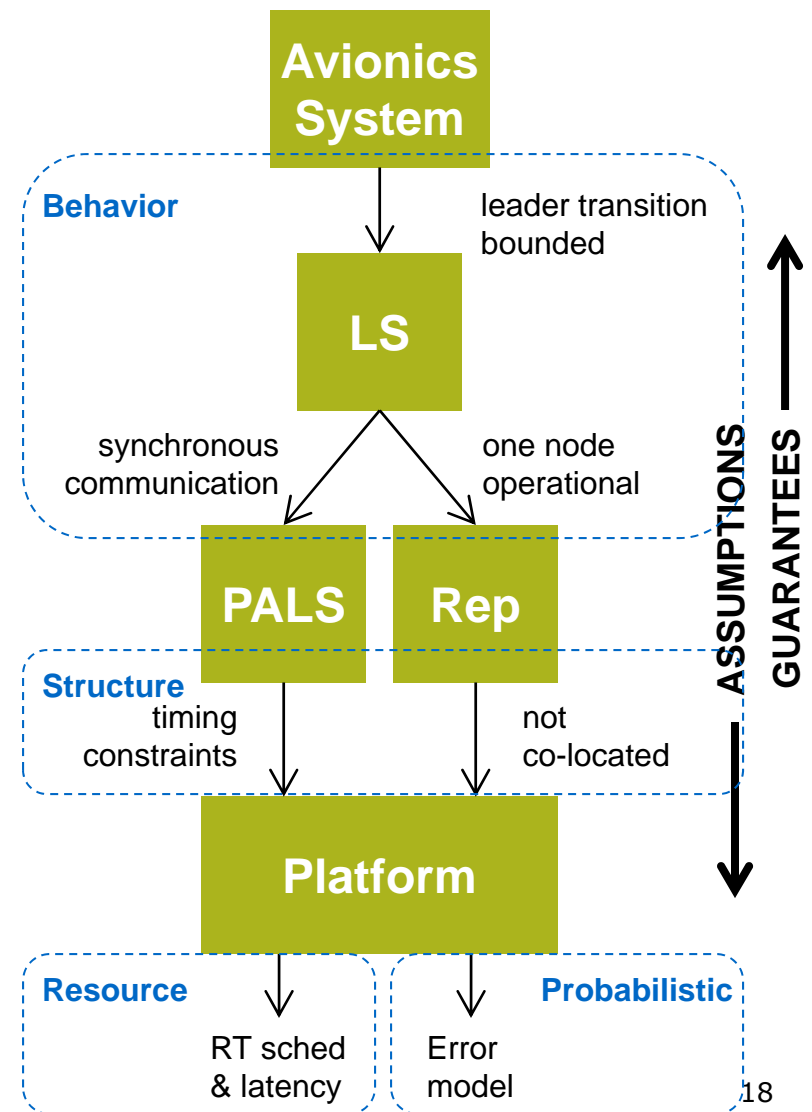
## Contracts between patterns and components

- Avionics system requirement

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

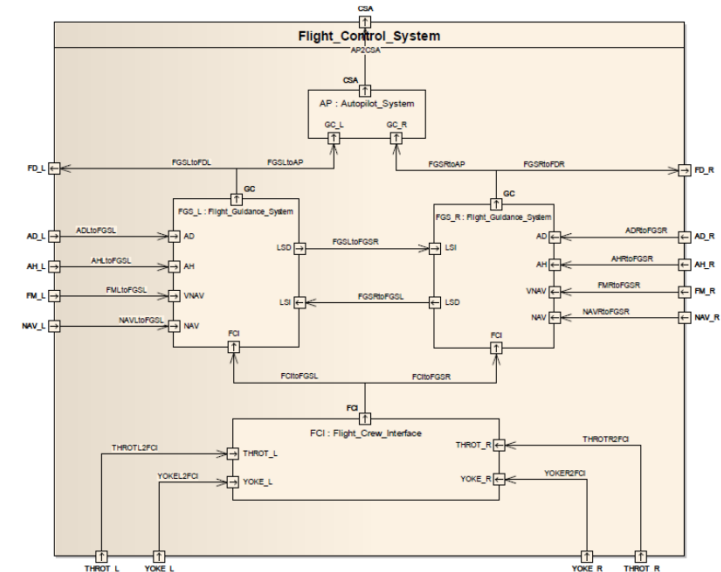
- Relies upon
  - Guarantees provided by patterns and components
  - Structural properties of model
  - Resource allocation feasibility
  - Probabilistic system-level failure characteristics

*Principled mechanism for “passing the buck”*

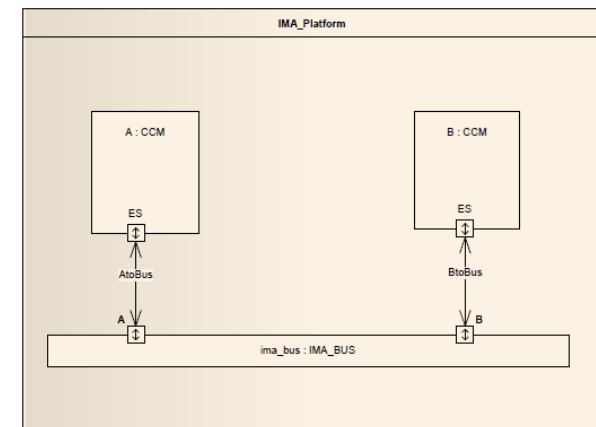


## Structural/static properties

- Software + HW platform
  - Process, thread, processors, bus
- Ex: PALS vertical contract
  - PALS timing constraints on platform
  - Check AADL structural properties
- Guarantees
  - Sync logic executes at PALS\_Period
  - Synchronous\_Communication => "One\_Step\_Delay"
- Assumptions (about platform)
  - Causality constraint:  
 $\text{Min}(\text{Output time}) \geq 2\epsilon - \mu_{\text{min}}$
  - PALS period constraint:  
 $\text{Max}(\text{Output time}) \leq T - \mu_{\text{max}} - 2\epsilon$



Software



Platform

## Structural property checks

- Contract
  - Platform model satisfies PALS assumptions
- Attached at pattern instantiation
  - Model-independent
  - Assumptions
  - Pre/post-conditions
- Lute theorems
  - Based on REAL
  - Eclipse plug-in
  - Structural properties in AADL model

```

PALS_Threads := {s in Thread_Set | Property_Exists(s,
"PALS_Properties::PALS_Id")};

PALS_Period(t) := Property(t, "PALS_Properties::PALS_Period");
PALS_Id(t) := Property(t, "PALS_Properties::PALS_Id");
PALS_Group(t) := {s in PALS_Threads | PALS_Id(t) = PALS_Id(s)};

Max_Thread_Jitter(Threads) :=
  Max({Property(p, "Clock_Jitter") for p in Processor_Set |
  Cardinal({t in Threads | Is_Bound_To(t, p)} > 0)});

Connections_Among(Set) :=
  {c in Connection_Set | Member(Owner(Source(c)), Set) and
  Member(Owner(Destination(c)), Set)};

theorem PALS_Period_is_Period
  foreach s in PALS_Threads do
    check Property_Exists(s, "Period") and
      PALS_Period(s) = Property(s, "Period");
  end;

theorem PALS_Causality
  foreach s in PALS_Threads do
    PALS_Group := PALS_Group(s);
    Clock_Jitter := Max_Thread_Jitter(PALS_Group);
    Min_Latency := Min({Lower(Property(c, "Latency")) for
      c in Connections_Among(PALS_Group)});
    Output_Delay := {Property(t, "Output_Delay") for t in PALS_Group};
    check (if 2 * Clock_Jitter > Min_Latency then
      Min(Output_Delay) > 2 * Clock_Jitter - Min_Latency
    else
      true);
  end;

```

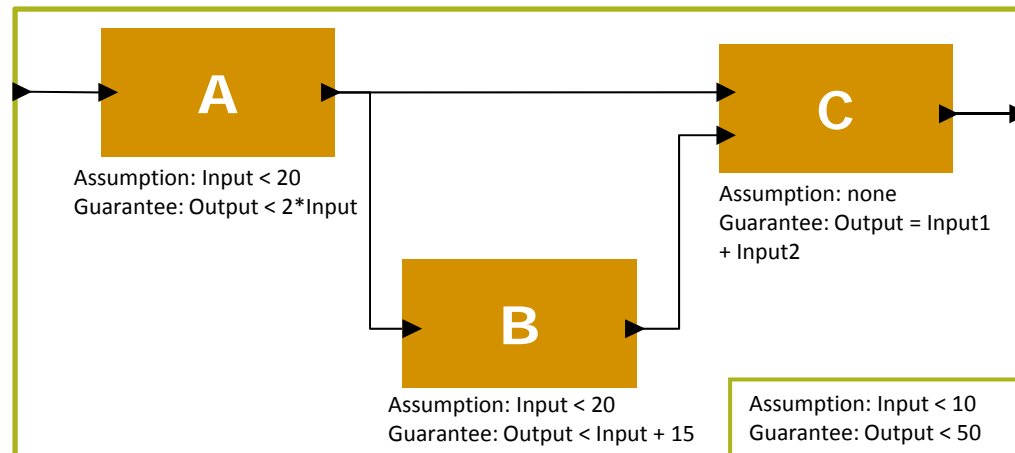
## Compositional behavior verification

- Given
  - Assumptions for system
  - Assumptions/Guarantees for components (A, P)
- Prove
  - System guarantees (requirements)
- New analysis plug-in (AGREE)
  - Automatic translation of model structure, contracts, and verification conditions
  - Verify via k-induction model checker (KIND - Tinelli @ Univ. of Iowa)

**Contract compliance:**  
 $G(H(A) \Rightarrow P)$

**Example (to prove)**

$$\begin{aligned}
 A_S &\rightarrow A_A \\
 A_S \wedge P_A &\rightarrow A_B \\
 A_S \wedge P_A \wedge P_B &\rightarrow A_C \\
 A_S \wedge P_A \wedge P_B \wedge P_C &\rightarrow P_S
 \end{aligned}$$



## Contract specification in AADL

- Derived from Property Specification Language (PSL) formalism
  - IEEE standard
  - In wide use for hardware verification
- Assume / Guarantee style specification
  - Assumptions: “Under these conditions”
  - Guarantees: “...the system will do X”
- Local definitions can be created to simplify properties
- For now, this is an AADL string property

```

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 ;
const CSA_MAX_PITCH_DELTA: real = 5.0 ;
const CSA_MAX_PITCH_DELTA_STEP: real = 5.0 ;

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

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

property Pitch_lr_ok =
  abs(AD_L.pitch.val - AD_R.pitch.val) < FCS_MAX_PITCH_SIDE_DELTA ;

property some_fgs_active =
  (FD_L.mds.active or FD_R.mds.active) ;

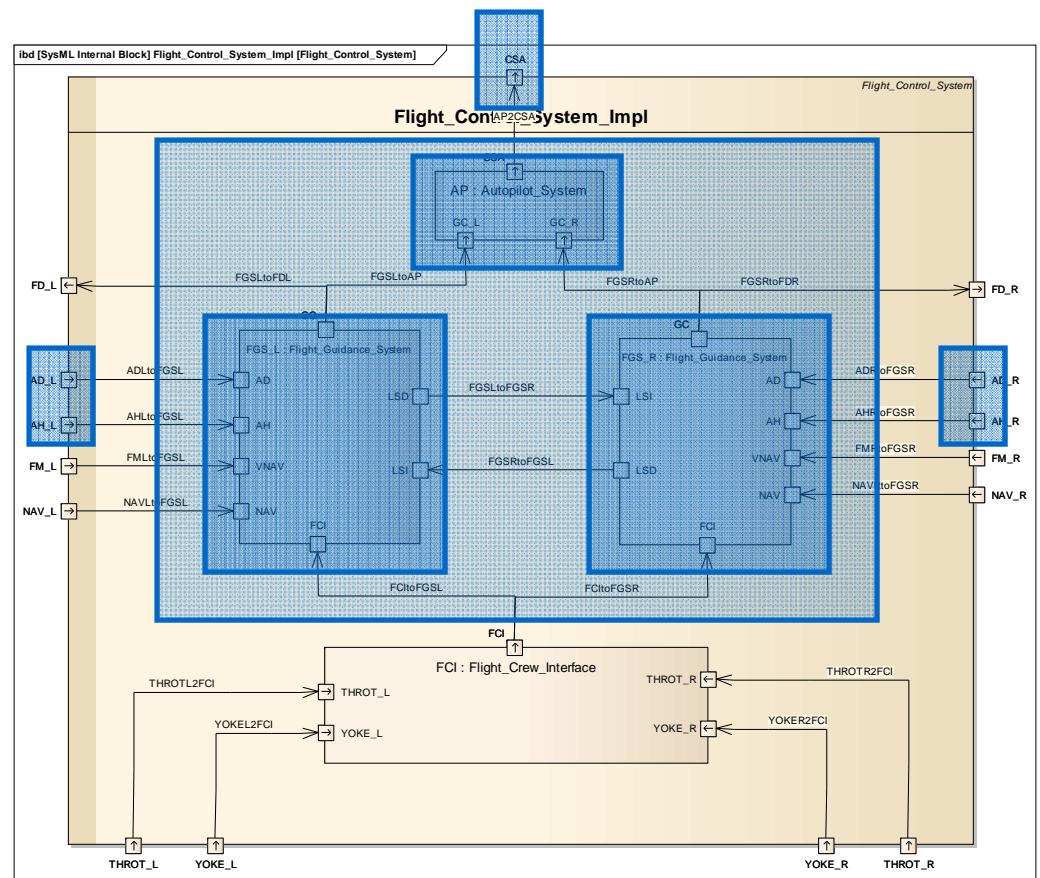
active_assumption: assume some_fgs_active ;

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 ;
  
```

## Compositional reasoning for FCS

- 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 ;
    
```

## Compositional Reasoning and Patterns

- Guarantees provided by pattern are encoded as **facts**
- Attached at pattern instantiation
  - Model-independent
  - Assumptions
  - Pre/post-conditions
- Describe relationships between several components
  - In this example, the *Leader* and *Valid* fields for the left and right FGSs.

```

pattern_instance Leader_Select_1 :

  -- sync single-step delay between elements
  assume single_step_delay_comm(FGS_L, FGS_R);
  assume single_step_delay_comm(FGS_R, FGS_L);

  -- All non-failed nodes agree on who is the leader
  leader_agreement:
    assert (FGS_L.LSO.Valid and FGS_R.LSO.Valid) =>
      FGS_L.LSO.Leader = FGS_R.LSO.Leader;

  -- If a node fails, leadership is transferred to a non-failed node
  leader_transfer_1:
    assert (prev(not(FGS_L.LSO.Valid), false) =>
      (FGS_R.LSO.Valid =>
        FGS_R.LSO.Leader != Get_Property(FGS_L, Leader_Select_ID)));

  leader_transfer_2:
    assert prev(not(FGS_R.LSO.Valid), false) =>
      (FGS_L.LSO.Valid =>
        FGS_L.LSO.Leader != Get_Property(FGS_R, Leader_Select_ID));

  -- If any non-failed nodes exist, one of them will be the leader
  leader_existence:
    assert (prev(FGS_L.LSO.Valid or FGS_R.LSO.Valid, false)) =>
      (( FGS_L.LSO.Valid => (FGS_L.LSO.Leader >= 1 and FGS_L.LSO.Leader <= 2)) and
       ( FGS_R.LSO.Valid => (FGS_R.LSO.Leader >= 1 and FGS_R.LSO.Leader <= 2)));

  -- If the leader does not fail, it shall remain the leader.
  leader_persistence_1: assert
    (prev(FGS_L.LSO.Valid and
      FGS_L.LSO.Leader = Get_Property(FGS_L, Leader_Select_ID), false)) =>
      (FGS_L.LSO.Valid =>
        FGS_L.LSO.Leader = Get_Property(FGS_L, Leader_Select_ID));

  leader_persistence_2: assert
    (prev(FGS_R.LSO.Valid and
      FGS_R.LSO.Leader = Get_Property(FGS_R, Leader_Select_ID), false)) =>
      (FGS_R.LSO.Valid =>
        FGS_R.LSO.Leader = Get_Property(FGS_R, Leader_Select_ID));

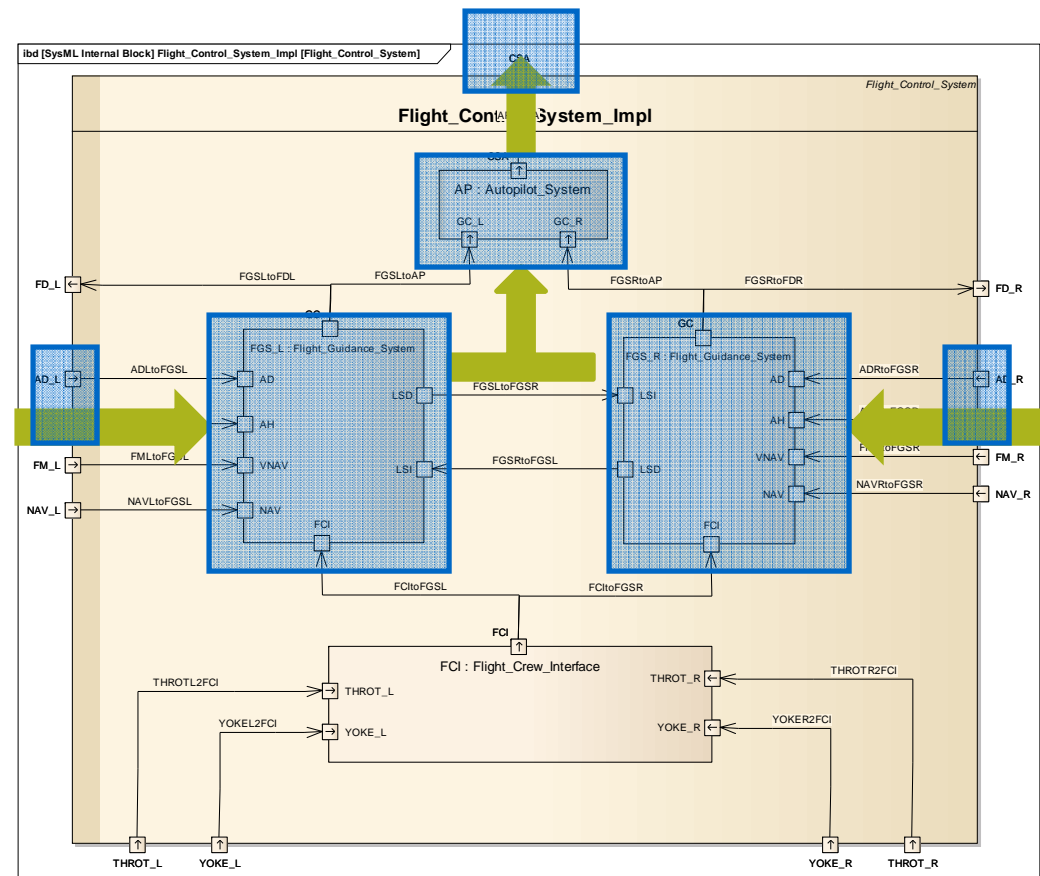
end pattern_instance Leader_Select_1 ;

```



## Proof Process

- Order of data flow through system components is computed by AGREE
  - {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 also handle circular flows, but user has to choose where to break cycle (usually a time delay)





# Verification tools

Lute

AGREE

Counterexample

The screenshot shows three instances of the Rockwell Collins AADL IDE. The rightmost instance is displaying a table titled 'System level guarantees' with columns A through K. The table contains 25 rows of data, including signal names, types, and numerical values.

	A	B	C	D	E	F	G	H	I	J	K
1	Signal	type	Step								
2				0	1	2	3	4	5		
3	AD_Latch_val	real		1.80952	0.85714	-0.09524	-1.04782	-1.66667	0.85714		
4	AD_Latch_valid	bool		FALSE	TRUE	FALSE	TRUE	TRUE	FALSE		
5	AD_R_pitch_val	real		3.47619	2.57143	1.61805	0.90476	-0.04762	-0.90476		
6	AD_R_pitch_valid	bool		TRUE	FALSE	TRUE	FALSE	FALSE	TRUE		
7	AP_CSA.cmds_pitch_delta	real		0	0.04762	3.42857	3.40476	3.38095	-1.61805		
8	AP_GC_L.cmds_pitch_delta	real		0	-0.04762	3.52381	-1.71429	-1.7619	-1.86687		
9	AP_GC_L.mds.active	bool		TRUE	FALSE	FALSE	FALSE	FALSE	TRUE		
10	AP_GC_R.cmds_pitch_delta	real		0	3.47619	2.52381	1.66667	0.85714	1.80952		
11	AP_GC_R.mds.active	bool		TRUE	TRUE	FALSE	FALSE	FALSE	FALSE		
12	Assumptions for AP	bool		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE		
13	Assumptions for FCI	bool		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE		
14	Assumptions for FGS_L	bool		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE		
15	Assumptions for FGS_R	bool		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE		
16	FGS_L_GC.cmds_pitch_delta	real		-0.04762	3.52381	-1.71429	-1.7619	-1.66667	-1.89048		
17	FGS_L_GC.mds.active	bool		FALSE	FALSE	FALSE	FALSE	TRUE	FALSE		
18	FGS_LLBO_leader	int		2	2	3	2	1	3		
19	FGS_LLBO_valid	bool		FALSE	TRUE	FALSE	TRUE	TRUE	FALSE		
20	FGS_R_GC.cmds_pitch_delta	real		3.47619	2.52381	1.66667	0.85714	1.80952	0.85714		
21	FGS_R_GC.mds.active	bool		TRUE	FALSE	FALSE	FALSE	FALSE	FALSE		
22	FGS_R_LBO_leader	int		0	0	1	0	1	1		
23	FGS_R_LBO_valid	bool		TRUE	FALSE	TRUE	FALSE	FALSE	TRUE		
24	leader_pitch_delta	real		0	3.47619	3.47619	3.47619	3.47619	-1.86687		
25	System level guarantees	bool		TRUE	TRUE	TRUE	TRUE	TRUE	FALSE		

## Next steps

- Extend compositional verification to more complex models of computation
  - Multiple rates, delays, asynchrony
- Incorporate additional design patterns in library
  - Especially fault tolerance patterns with existing verification artifacts
- Improved annotation of contracts in architecture models
  - AADL annex? Alternate representations (e.g., sequence diagrams?)
- More general mechanism for composing evidence from multiple sources
  - Evidence graph, assurance case

## Download

- AADL Tools wiki
  - [https://wiki.sei.cmu.edu/aadl/index.php/RC\\_META](https://wiki.sei.cmu.edu/aadl/index.php/RC_META)

