



# Compositional Reasoning for Architectural Models

Software Certification Consortium

9 May 2016

Darren Cofer  
Advanced Technology Center  
Trusted Systems

John Backes  
Andrew Gacek  
Mike Whalen (UMN)



# Verification of Safety-Critical Embedded Software

- **Problem**

- As complexity increases, verification has become the most costly part of development
- Integration can be especially challenging and expensive

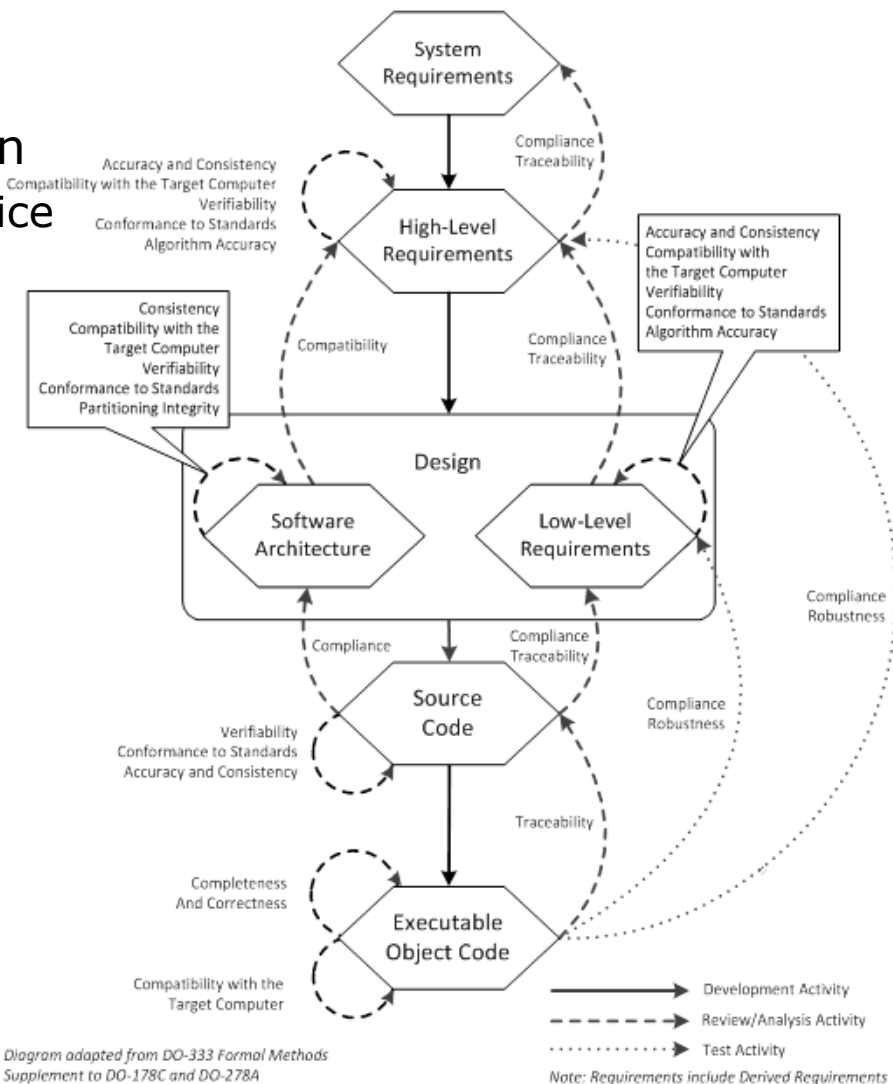
- **Approach**

- Model system architecture with standard notation with clear semantics
- Divide verification task into manageable, reusable pieces
- Translate these models to a form that can be interpreted by powerful general-purpose engines (SMT-based model checkers)

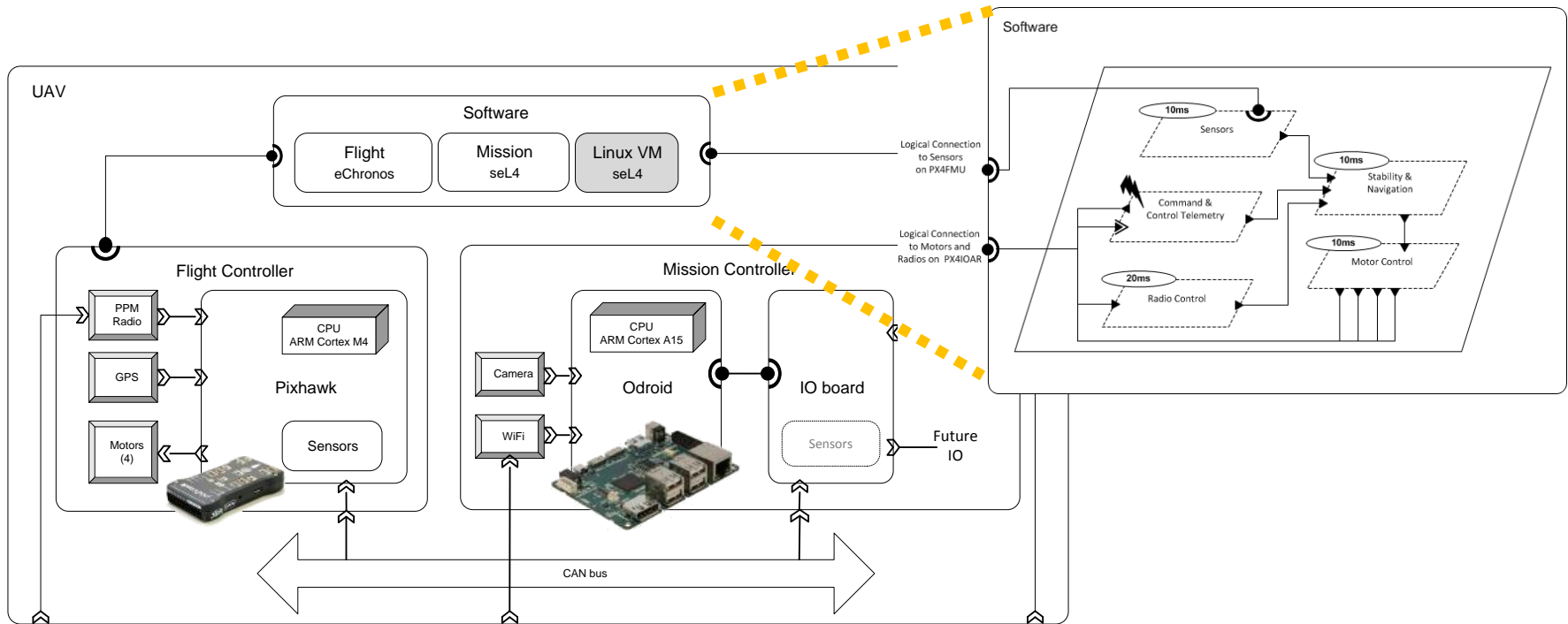


# Certification

- We also care about certification
  - Compliance with current practice and certification guidance



# Architecture Analysis and Design Language (AADL)



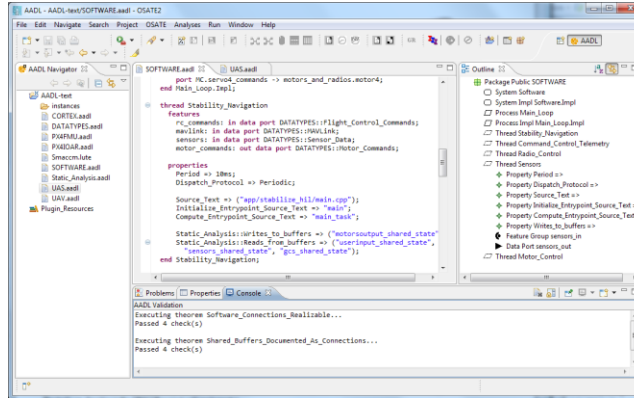
## AADL = SAE AS5506 standard

- Target: Embedded, real-time, distributed systems
- Describes both hardware and software
- Extensible syntax (annex)
- Open source tools, supported by SEI

# Architecture Modeling and Analysis Tools

**OSATE**

**Trusted Build**



**Resolute Assurance Case**

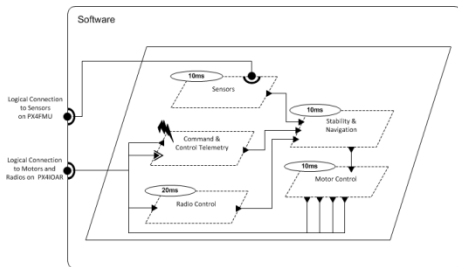
**AGREE Behavioral Analysis**

**Sute Structural Analysis**

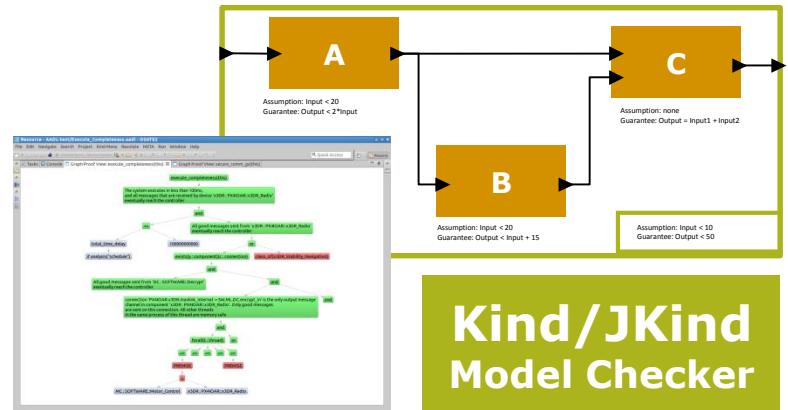
Architecture Models

Architecture Translation

Architecture Analysis



**seL4**  
**eChronos**  
**VxWorks**

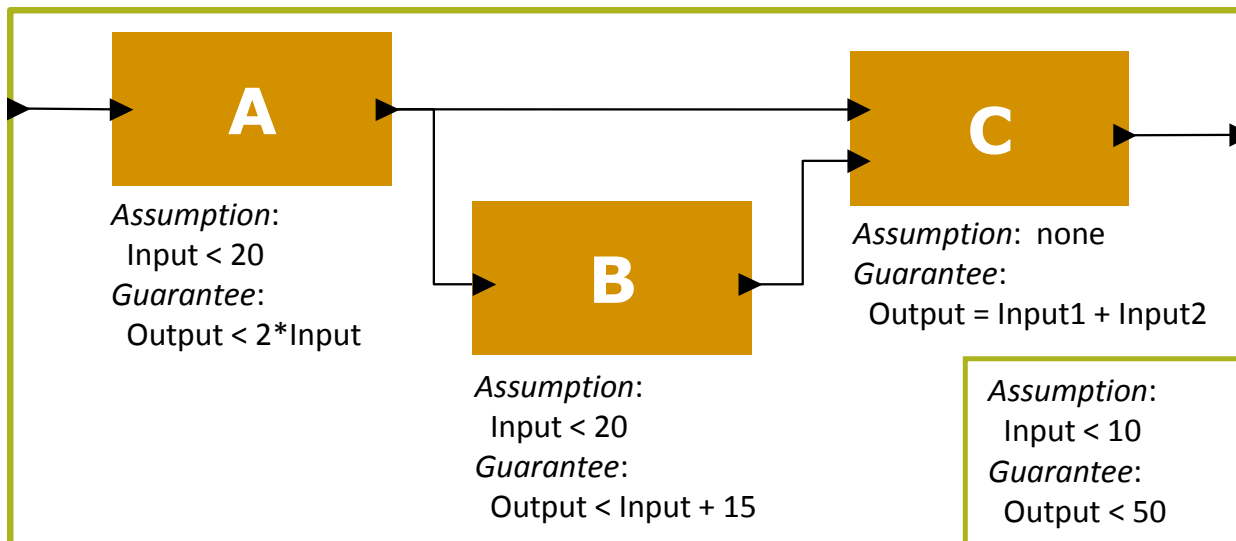


**Kind/JKind Model Checker**

## AGREE: Assume-Guarantee Reasoning Environment

- Each AADL component contains a **assumptions** about its inputs and **guarantees** about its outputs in a **contract**.
- AGREE uses a k-induction model checker to determine whether a component implementation (its subcomponents) satisfies its contract.
- Analysis proceeds hierarchically on the model:
  - Evidence that a component's guarantees hold only relies on evidence from its subcomponents.
  - This allows analysis to scale to larger designs.

## Simple Example



Assuming the **input** to the system <10, is the **output** <50?

- Given
  - System assumptions
  - Subcomponent contracts
  - Architecture model
- Prove
  - System guarantees

### Proof Obligations

$$A_S \rightarrow A_A$$

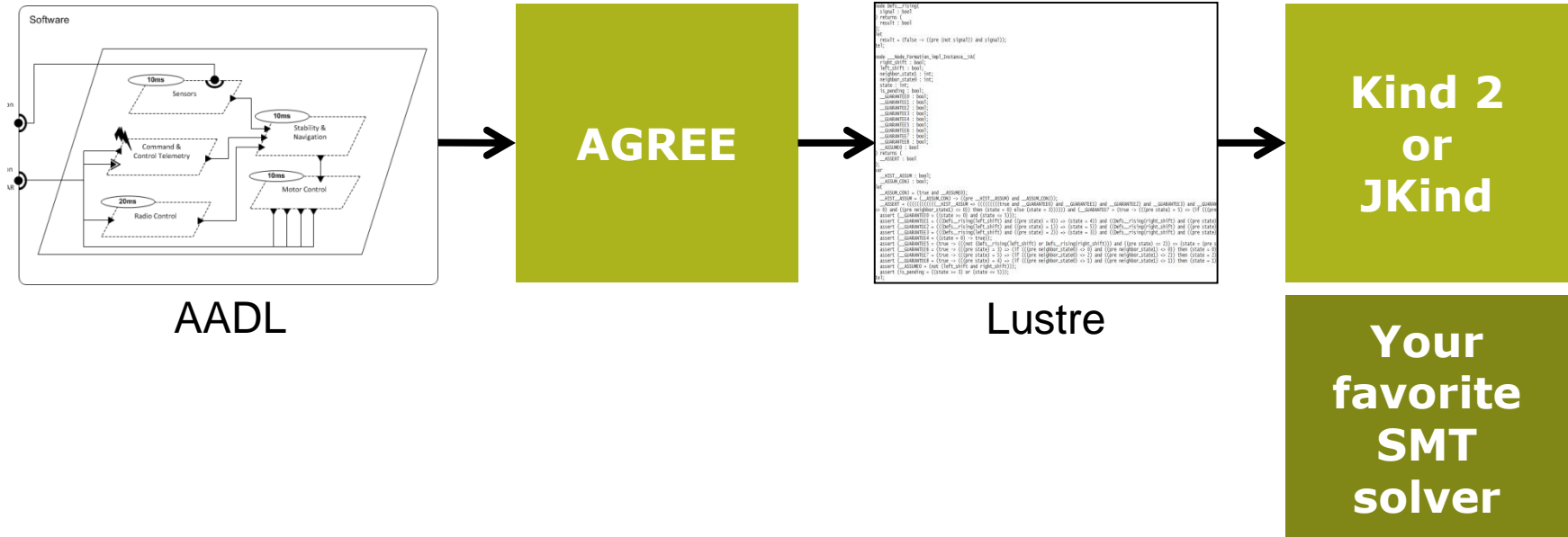
$$A_S \wedge G_A \rightarrow A_B$$

$$A_S \wedge G_A \wedge G_B \rightarrow A_C$$

$$A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$$

# AGREE Translation

- Translates AADL models with AGREE annotations into Lustre for analysis by k-induction model checkers
  - Kind 2 (University of Iowa)
  - JKind (Rockwell Collins)





## AGREE Annex to AADL

- Annotate AADL model with assume-guarantee contracts

```
system Car
  features
    Target_Speed: in data port Types::speed.speed_impl;
    Actual_Speed: out data port Types::speed.speed_impl;

  annex agree {**

    const MAX_ACCEL : real = 2.0;

    assume "target speed is positive" : Target_Speed.val >= 0.0;
    assume "reasonable target speed" : Target_Speed.val < 150.0;

    eq const_tar_speed : bool =
      Agree_Nodes.H(Target_Speed.val = prev(Target_Speed.val, 0.0));

    guarantee "actual speed is less than constant target speed" :
      const_tar_speed => (Actual_Speed.val <= Target_Speed.val);

    guarantee "acceleration is limited" :
      Agree_Nodes.abs(Actual_Speed.val - prev(Actual_Speed.val, 0.0)) < MAX_ACCEL;

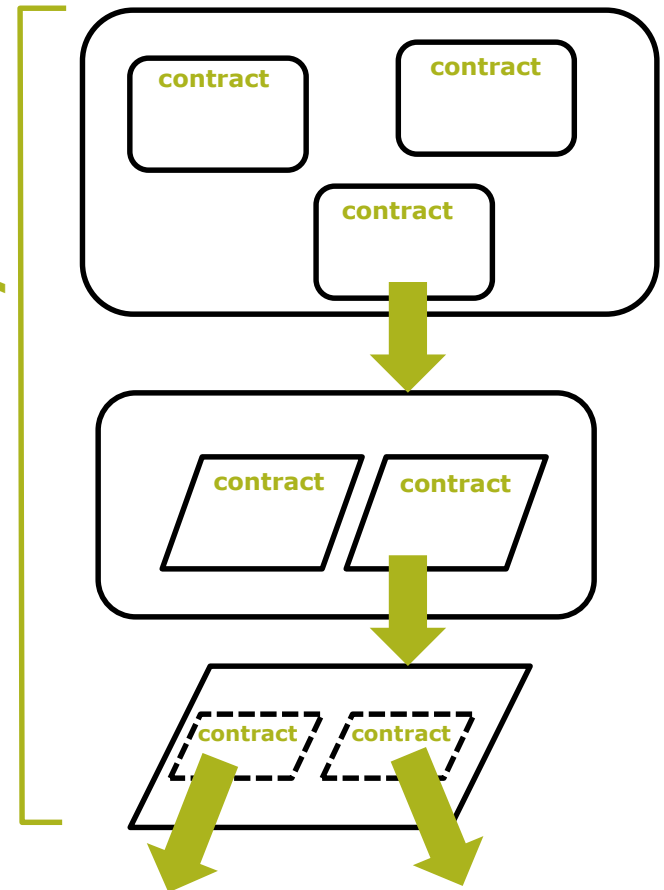
  **};

end Car;
```

# Export component contracts to Simulink

- Formal contracts on components
  - Assumptions: Constraints on what a component expects from its environment
  - Guarantees: Specification of component behavior in response to its environment
- The contract of a component specifies requirements for its *implementation*
- Tools link system development environment to component development environment

AGREE analysis



Component Implementation

```

/*
 * File: heading_control.c
 * Code generated for Simulink model 'heading_control'.
 */

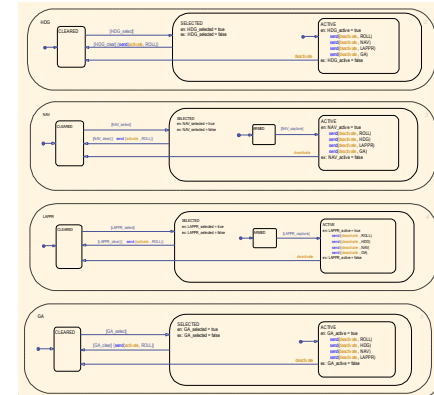
#include "heading_control.h"
#include "heading_control_private.h"

/* Real-time model */
RT_MODEL_heading_control heading_control_M;
RT_MODEL_heading_control *const heading_control_M = &heading_control_M;

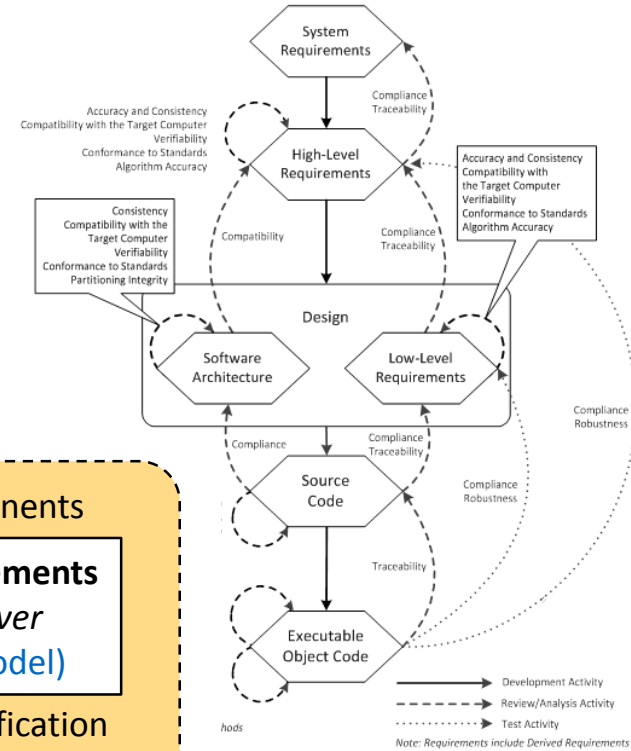
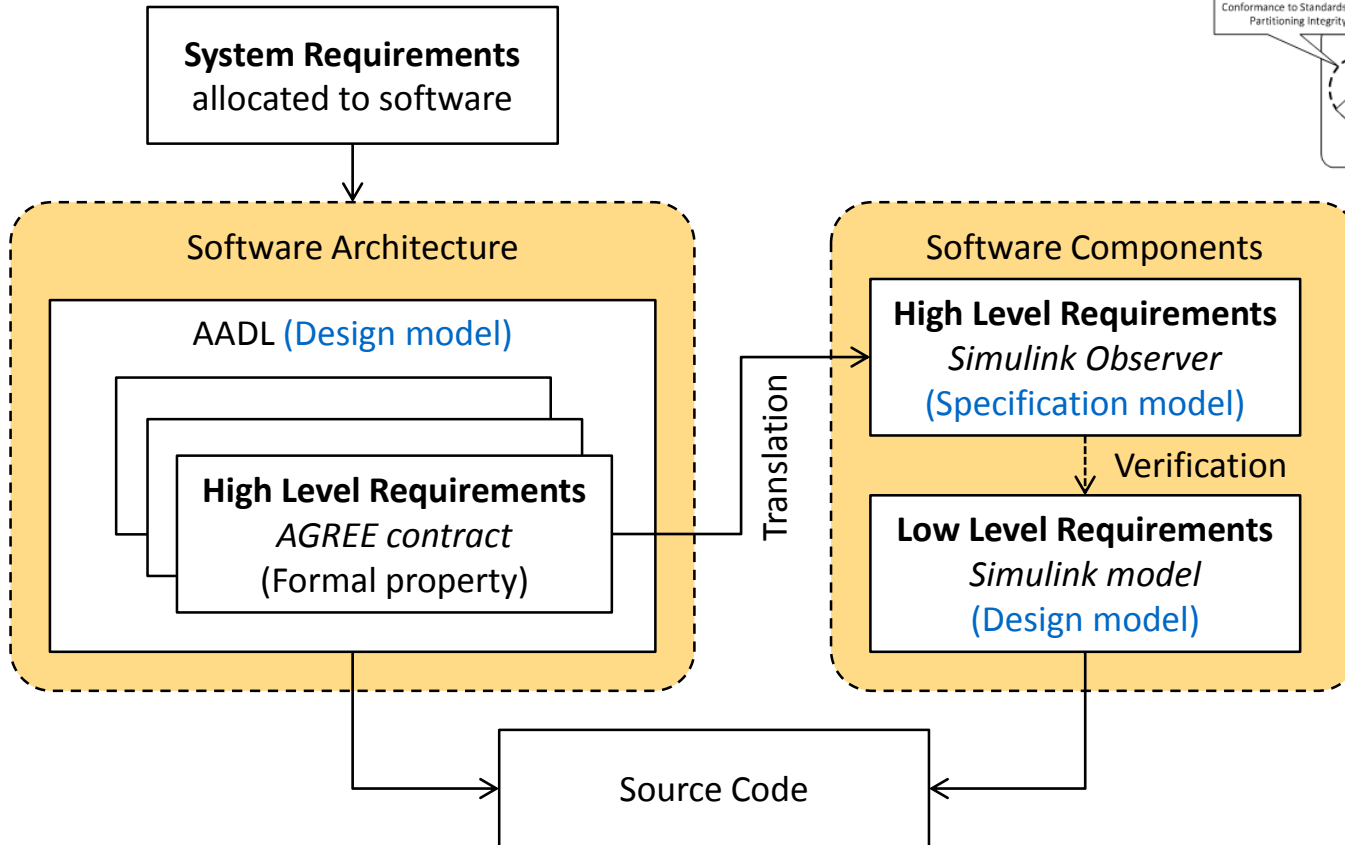
/* Model step function */
void heading_control_step(void)
{
    real_T denAccum;
    real_T u;
    real_T u_0;

    /* DiscreteIntegrator: '<S3>/Discrete-Time Integrator' */
    heading_control_B.DiscreteTimeIntegrator =
        heading_control_DWork.DiscreteTimeIntegrator_DSTATE;

    /* Gain: '<S3>/Gain1' */
    heading_control_B.Gain1 = 0.04 * heading_control_B.Discre
    
```



# DO-178C Process



# Export Design Contracts

The screenshot shows the Eclipse IDE with the AADL editor open to 'Integer\_Toy.aadl'. The code defines three systems (A, B, and C) with features and an 'annex agree' block. A context menu is open over the 'Integer\_Toy' package, showing options for verification and export. A second context menu is open over the 'System Implementation A.i' node in the Outline view, showing options for debugging and saving as XML.

```

package Integer_Toy
public
  with Base_Types;

  system A
  features
    Input: in data port Base_Types::Integer;
    Output: out data port Base_Types::Integer;

  annex agree {**
    assume "A input range" : Input < 20;
    guarantee "A output range" : Output < 2*Input;
  **};
end A ;

  system implementation A.i
  end A.i;

  system B
  features
    Input: in data port Base_Types::Integer;
    Output: out data port Base_Types::Integer;

  annex agree {**
    assume "B input range" : Input < 20;
    guarantee "B output range" : Output < Input + 15;
  **};
end B ;

  system C
  features
    Input1: in data port Base Types::Integer;
  
```

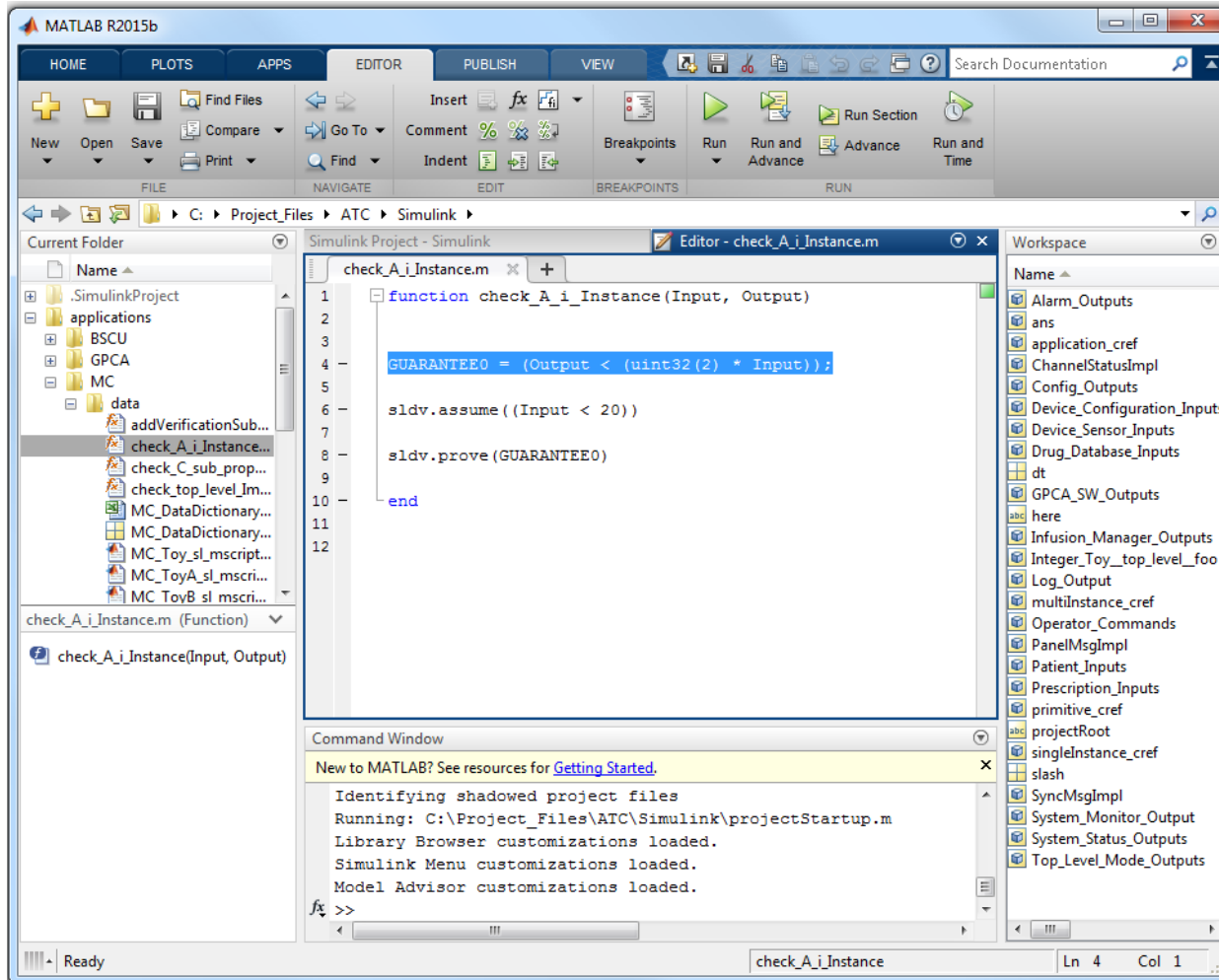
Context menu options (over Integer\_Toy package):

- Verify Single Layer
- Verify All Layers
- Check Realizability
- Verify Monolithically
- Generate MATLAB Function Single Layer

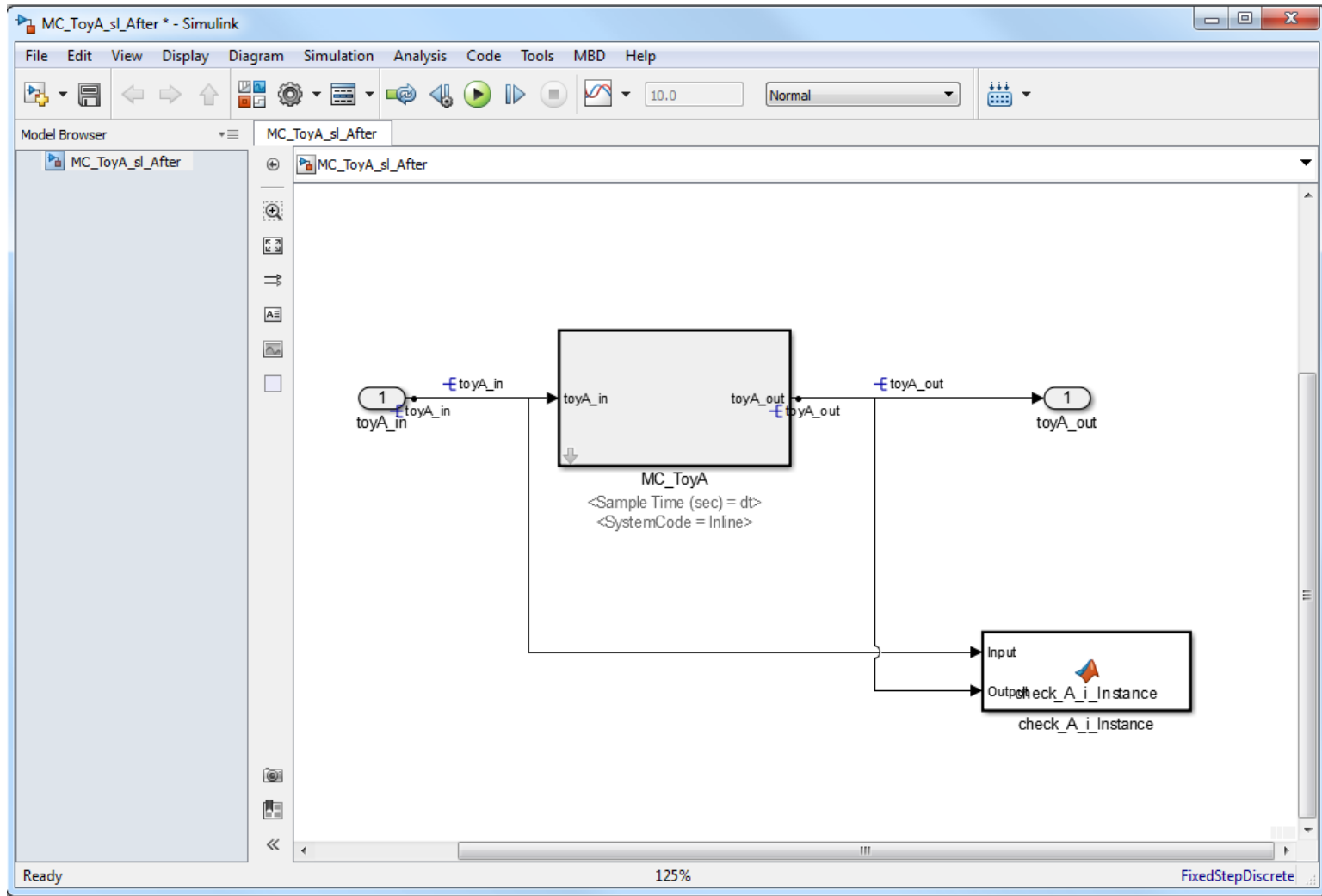
Context menu options (over System Implementation A.i):

- Debugging: Serialize as 'mypack.aadl' text
- Save As XML
- Instantiate System
- Ocarina
- Open Diagram
- Resolute
- Predefined Theorems
- AGREE

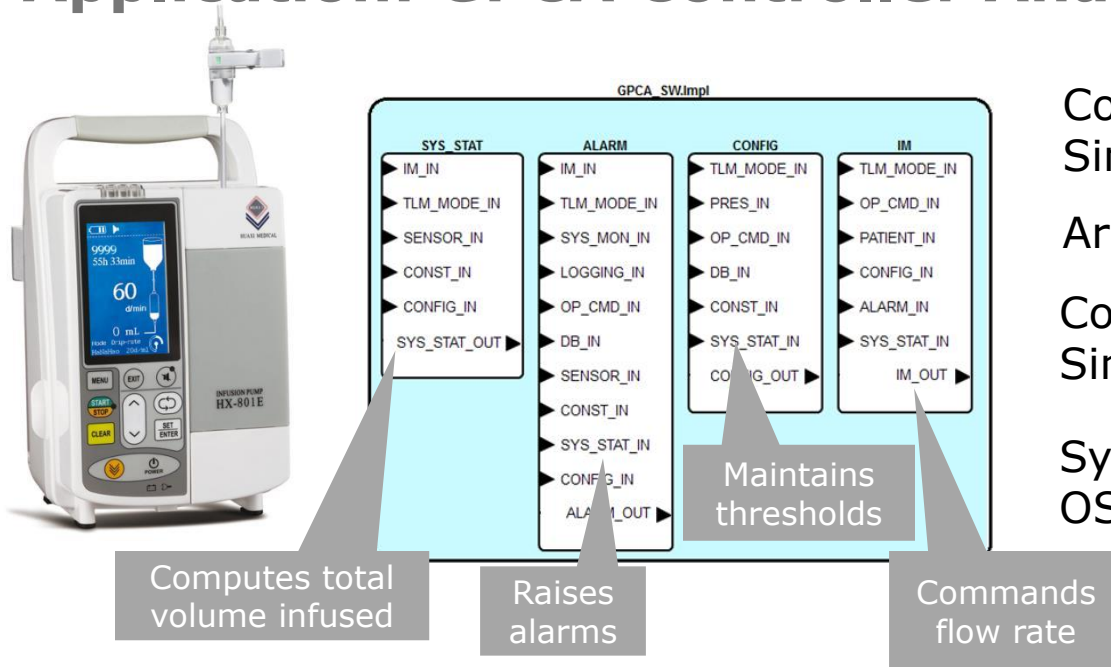
# Translate to MATLAB Function



# Insert Property Observer in Simulink Model



# Application: GPCA Controller Analysis



Components modeled in Simulink

Architecture modeled in AADL

Component verification using Simulink Design Verifier

System verification using OSATE/AGREE

## Example Requirement:

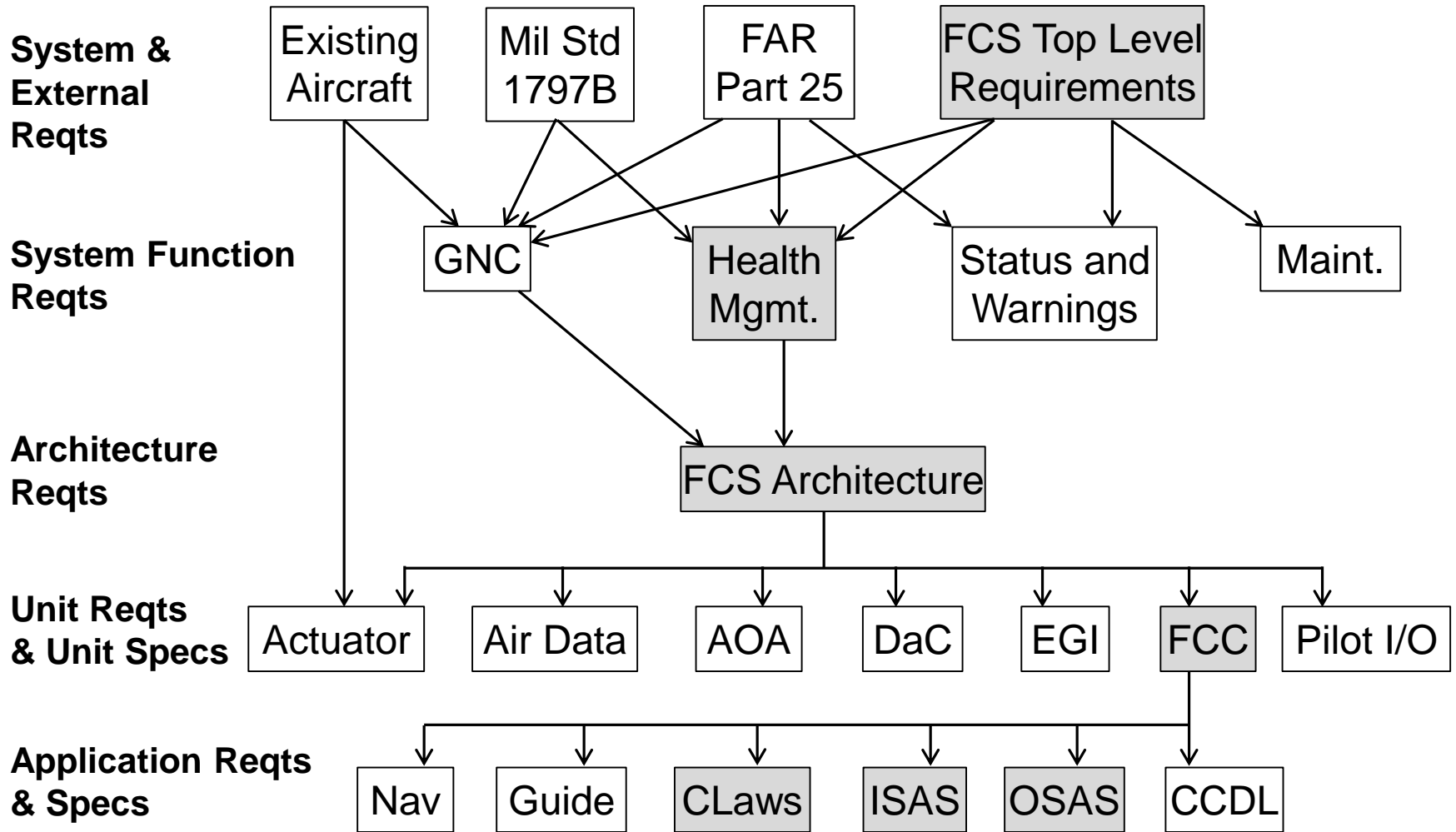
**If the drug reservoir is empty, the infusion flow rate shall be zero.**

**Counterexample found in <1 second**

## Analysis Time:

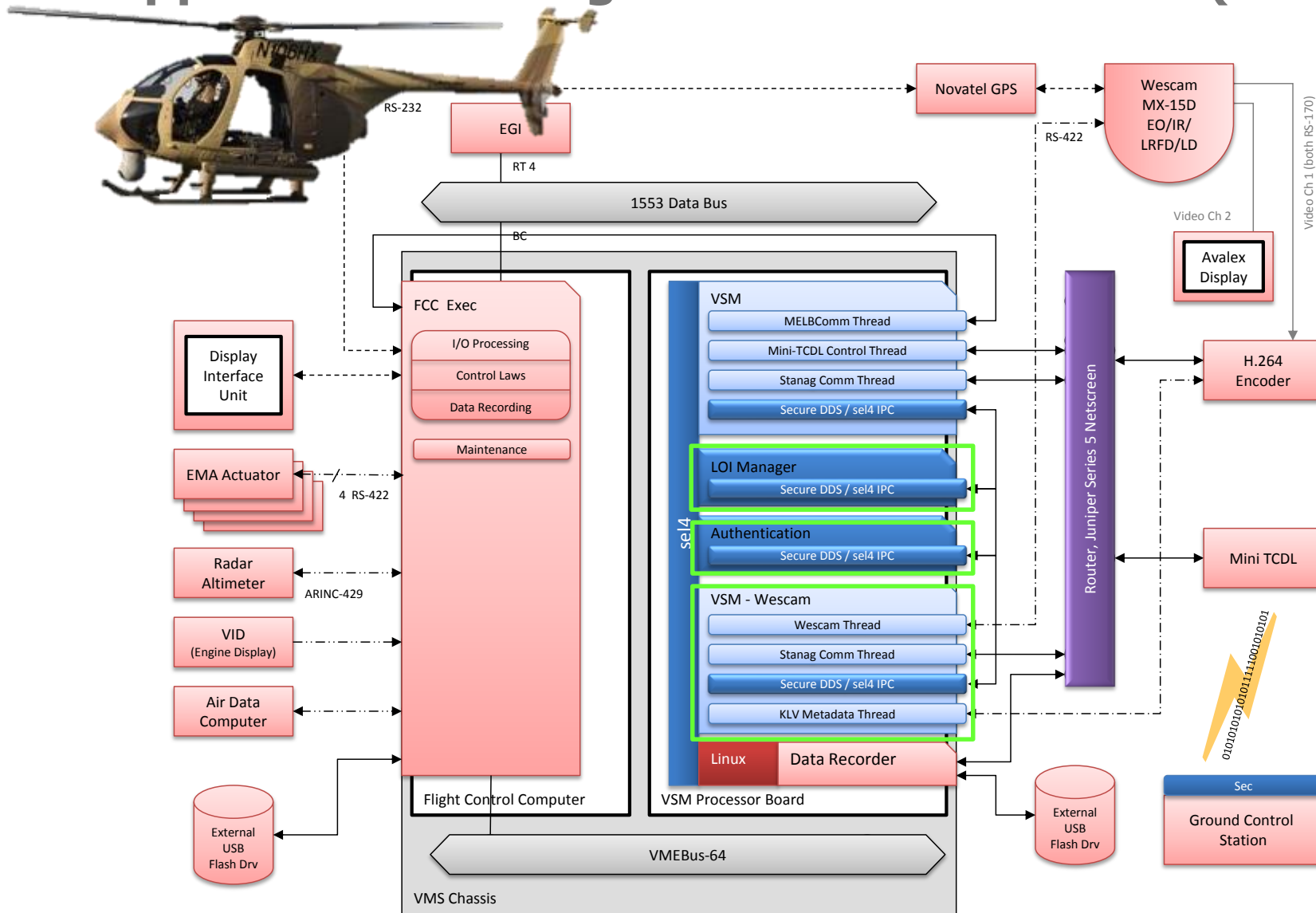
Monolithic Model	Composed Model
> 1 hour (12/19 proved)	4m 33s

# Application: NASA Quad-Redundant Flight Control System

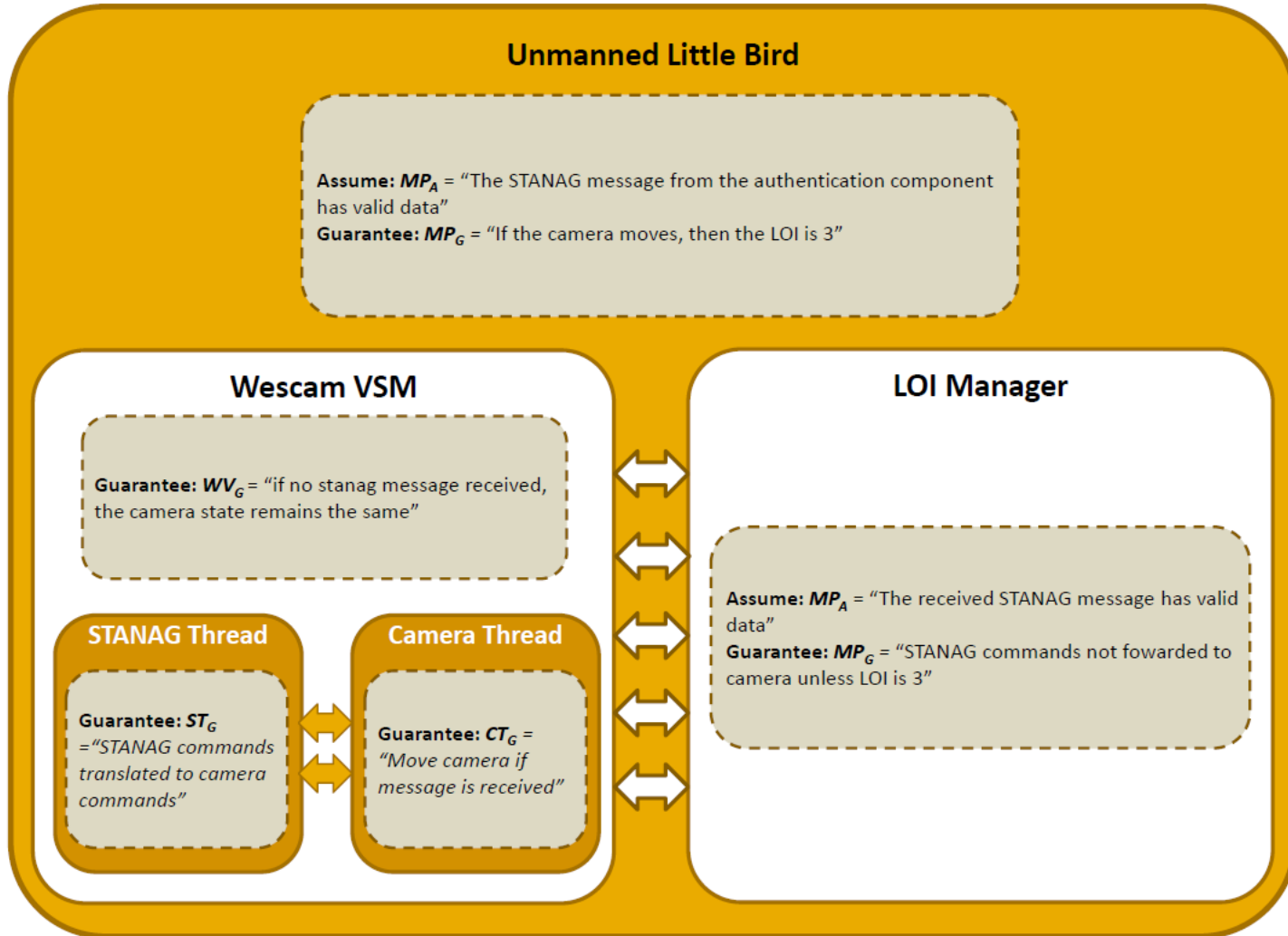




# Application: Boeing Unmanned Little Bird (HACMS)



# Example: ULB Contracts



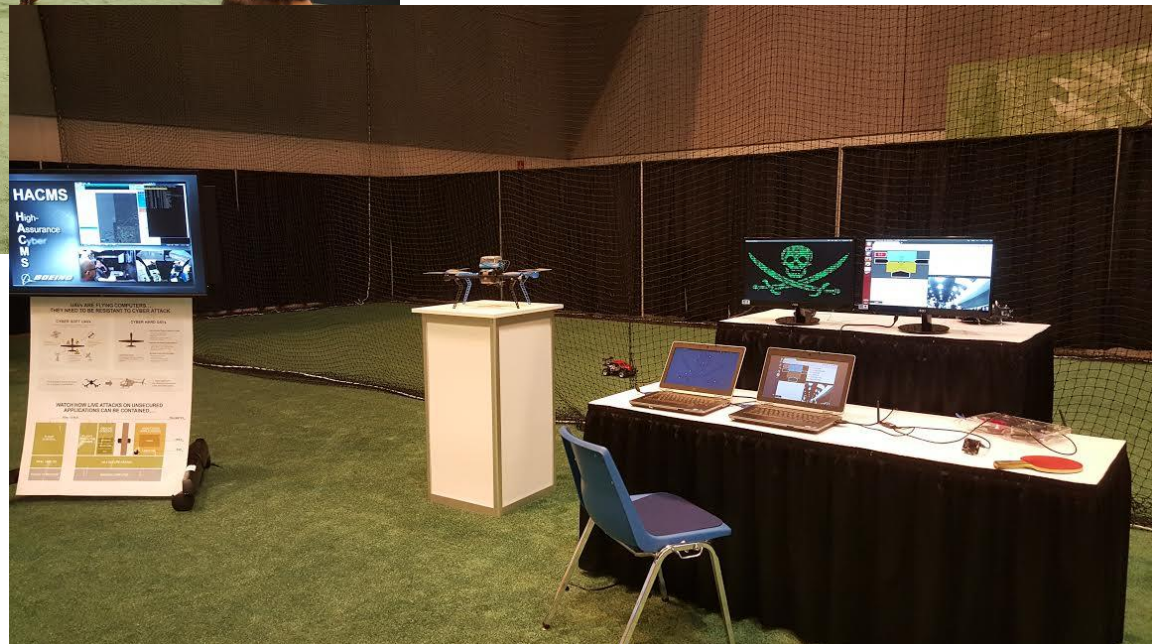
## ULB Phase 2 Flight Demo – 24 July, Mesa AZ



# DARPA "Wait, What?" St. Louis, 9-11 Sept 2015



Formal  
Methods



## Observations

- Some requirements are not compositional
  - High level requirements contain details of low level specifications
- Errors found during formalization
  - Problems became evident when translating the English requirements to AGREE syntax
- Errors found during model checking
  - Counterexample provided by Kind 2/JKind
- Errors found by realizability analysis
  - No possible implementation existed for a component with the given requirements

## Current Work

- Support for more AADL constructs
  - Shared Data
  - Remote Procedure Calls
- Exporting component contracts to other environments
  - C/C++
- Changes to specification language
  - More expressive, more user-friendly
- Richer timing semantics
  - Real-time

# Loonwerks

More information, code, and papers available at:

**Loonwerks.com**

