International Council on Systems Engineering
A better world through a systems approach

Creating Better System Models:
A Method for Using Compositional Reasoning to Validate Architectures with Assumption/Guarantee Contracts

MathWorks: Josh Kahn, Vidya Srinivasan

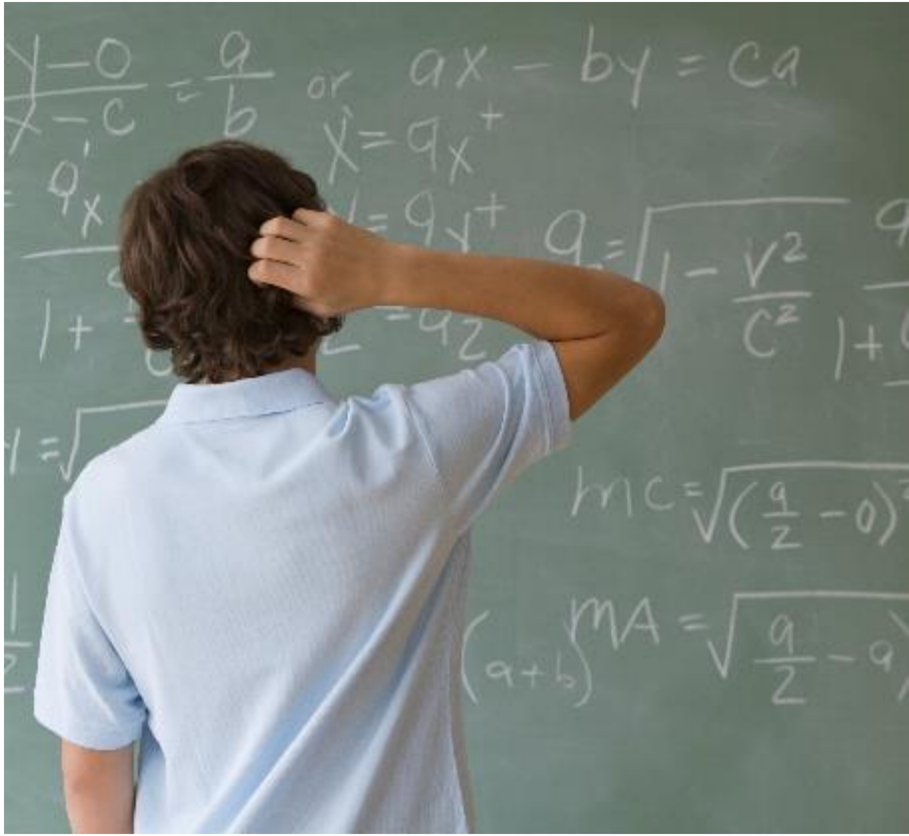
Collins Aerospace: Isaac Amundson, Gopal Narayan Rai, Janet Liu

INCOSE International Symposium 2025 | Ottawa, Canada

The Motivation

Formal methods have proved to be a valuable tool for **early identification** of defects in **safety-critical systems** so why aren’t they being broadly used in the systems engineering community?

- Lack of Commercial Tools
- Poor Integration with Existing MBSE Tools
- Cryptic Results



Why this Matters

Integration Issues Happen All. The. Time.

Sometimes they are caught during integration testing, and sometimes…

- Patriot Defense System – Inaccurate Tracking System[1]
- Mars Climate Orbiter – Data Unit Mismatch[2]
- Three Mile Island – Indicator Lights Based on Command, Not Feedback[3]
- Boeing 737 Max – MCAS Reliance on a Single AOA Sensor[4]

1. PATRIOT MISSILE DEFENSE: Software Problem Led to System Failure at Dhahran, Saudi Arabia, https://apps.dtic.mil/sti/citations/ADA344865
2. Mars Program Independent Assessment Team Report, https://ntrs.nasa.gov/citations/20000032458
3. Three Mile Island Accident, https://world-nuclear.org/information-library/safety-and-security/safety-of-plants/three-mile-island-accident
4. Summary of the FAA's Review of the Boeing 737 MAX, https://www.faa.gov/sites/faa.gov/files/2022-08/737_RTS_Summary.pdf

“It is often the case that many of the errors in system development manifest themselves in integration; each of the leaf-level components meets its requirements, but these are not sufficient to establish the satisfaction of the system requirements.”

Whalen et al., 2013

1. Your “What” Is My “How”: Iteration and Hierarchy in System Designm, Whalen et al, https://doi.org/10.1109/MS.2012.173

Hello.

Josh Kahn

Principal Systems Engineering Strategist

- Customer Problem Solving
- Industry Engagement and Feedback
- Strategic Direction Setting
- Internal Leadership and Guidance

joshkahn@mathworks.com

linkedin.com/in/josh-kahn-mbse

2009 – B.S Mechanical Engineering

2011 – M.Eng Space Systems Engineering

2010 – Project Manager, Test Engineer

2012 – System Integration Product Team Lead

2015 – Advanced Lead System Engineer

2020

2022 – Best in Conference

Today's Agenda

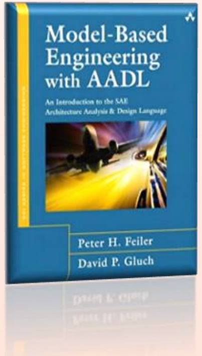
- AADL and AGREE:
  The Blueprint
- Enabling Broader Adoption
  with Commercially-Available Tools
- Making Sense of the Data
  Creating Actionable Results
- Where Do We Go from Here?
  Key Takeaways & Next Steps

AADL and AGREE

The Blueprint

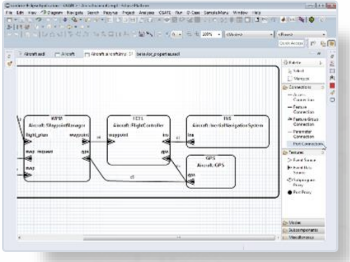
```
package Aircraft
public
    with CASE_Props;
```

Architecture Analysis and Design Language (AADL)



Textual and graphical language for modeling embedded, real-time, distributed systems

SAE AS5506



Open Source Tooling Supported by Carnegie Mellon Software Engineering Institute (CMU SEI)

Open **S**ource **A**ADL **T**ool **E**nvironment (OSATE)

Basic Building Blocks of the Language

Physical Hardware

- processor
- bus
- memory
- device

Application Software

- process
- thread
- subprogram
- data

Rigorous Semantics for Formal Analysis

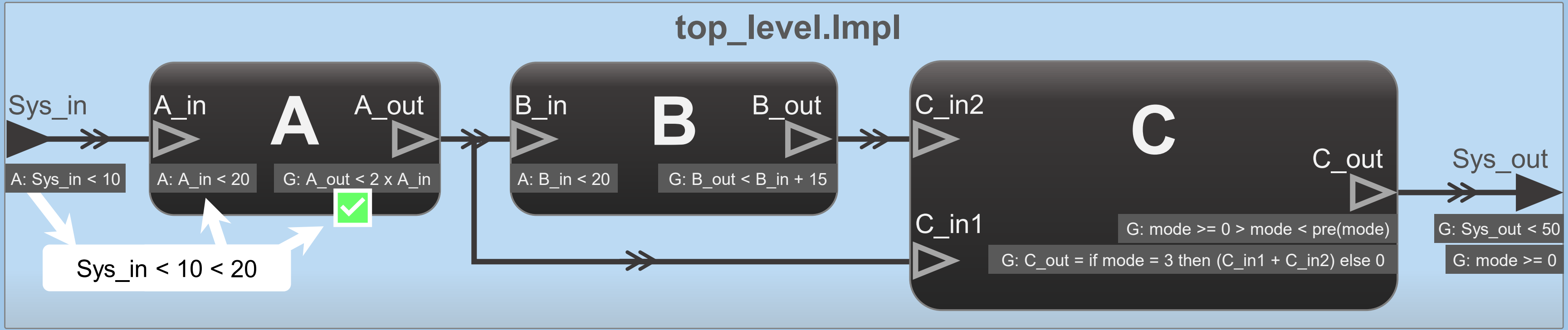
Extendable Syntax (Annexes)

Planned Support for **SysML v2**

```
end WaypointManager;
```

Compositional Reasoning with AGREE

Assume **G**uarantee **Re**asoning **E**nvironment



To prove correctness of

- ✓ **Component Interfaces**
  component assumptions are satisfied by upstream guarantees
- ✓ **Component Implementations**
  component assumptions and subcomponent guarantees satisfy guarantees

Assumptions describe the expectations that a component has on the environment

Guarantees describe bounds on the behavior of the component when assumptions are valid

Enabling Broader Adoption

with Commercially-Available Tools

Things We Needed

…to address barriers to adoption of existing formal methods tools

Commercially Available Tool(s)

- IT departments shy away from open-source
- Homegrown tools require local expertise and upkeep/support
- Non-commercial options have limited support, examples, and documentation

An Architecture Modeling Tool

- Model Architectures of Systems
- Associate AGREE-style contracts with them
- Graphical Editing

An Analytical Engine

- Property Proving Capability
- Reduce complexity
- Crunch the numbers
- Results Visualization

Interoperability and Extensibility

The Stack

the tools we chose to implement our proof-of-concept

MATLAB®

- most engineers already have it
- well-supported with public doc and examples
- powerful
- toolboxes

System Composer™

- intuitive architecture modeling and diagramming
- profiles and stereotypes for extensibility
- API access

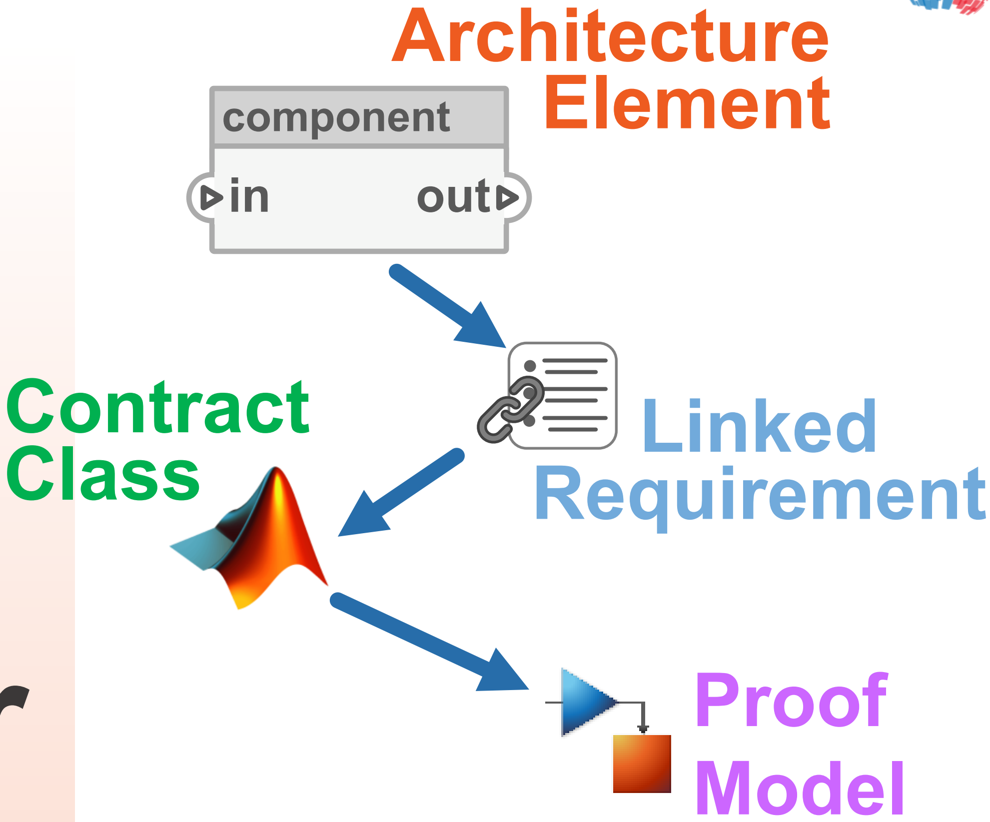
Requirements Toolbox™

- assume/guarantee contracts as verifiable requirements
- native integration with MATLAB and System Composer

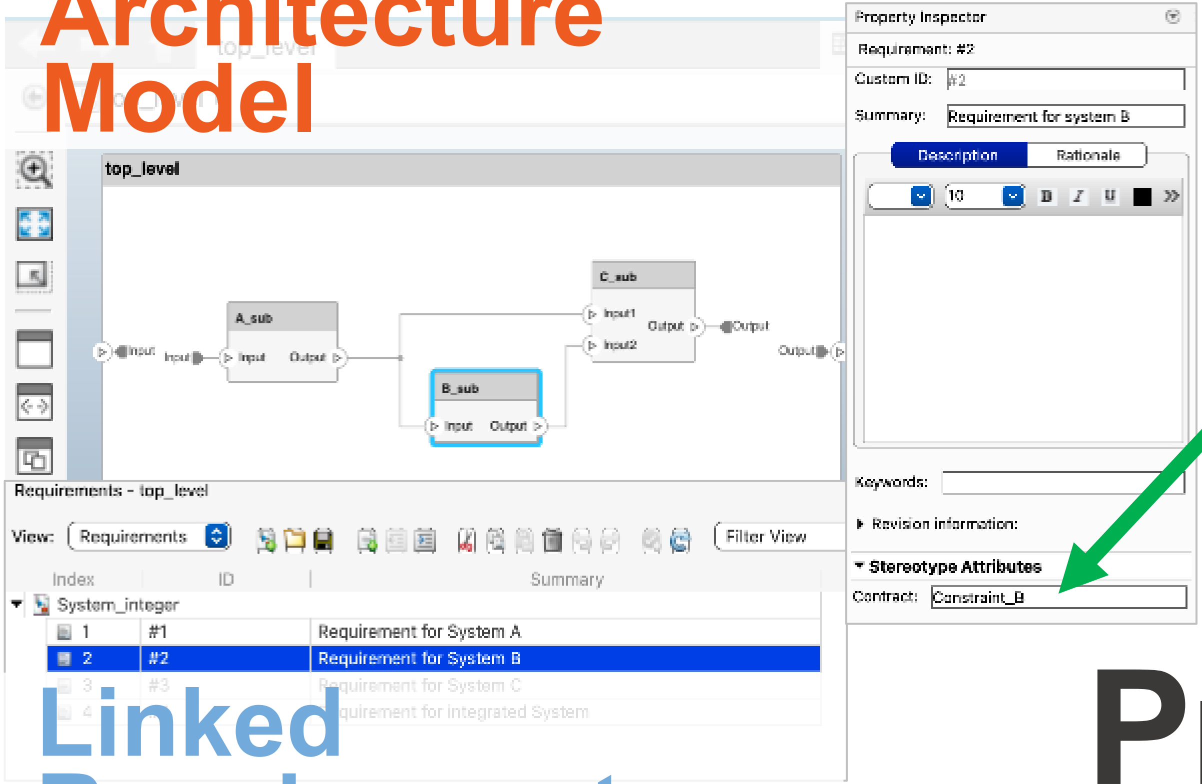
Simulink Design Verifier™

- mature formal methods tool
- native integration with MATLAB and System Composer

Putting It All Together



Architecture Model



Assume / Guarantee Contract

In Practice

Linked Requirements

The Contract

Using a generalized MATLAB class for the contract gave us

- syntax highlighting
- linting
- reusability
- access to other toolboxes

```
classdef Constraint_B < agree.AbstractContstraint
    % This class defines the AGREE contract for System B

    methods
        function this = Constraint_B()
            this.Description = 'Constraint for system B’;
        end
    end

    methods
        function tf =  getAssumption(~, Input)
            tf = Input < int32(20);
        end

        function tf = getGuarantee(~, Input, Output)
            tf = Output < Input + int32(15);
        end
    end
end
```

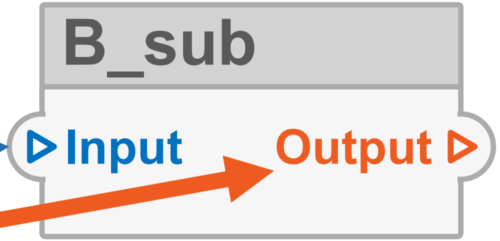
```
classdef Constraint_B < agree.AbstractContstraint
    % This class defines the AGREE contract for System B

    methods
        function this = Constraint_B()
            this.Description = 'Constraint for system B';
        end
    end

    methods
        function tf =  getAssumption(~, Input)
            tf = Input < int32(20);
        end

        function tf = getGuarantee(~, Input, Output)
            tf = Output < Input + int32(15);
        end
    end
end
```

We correlated class method arguments to ports by name



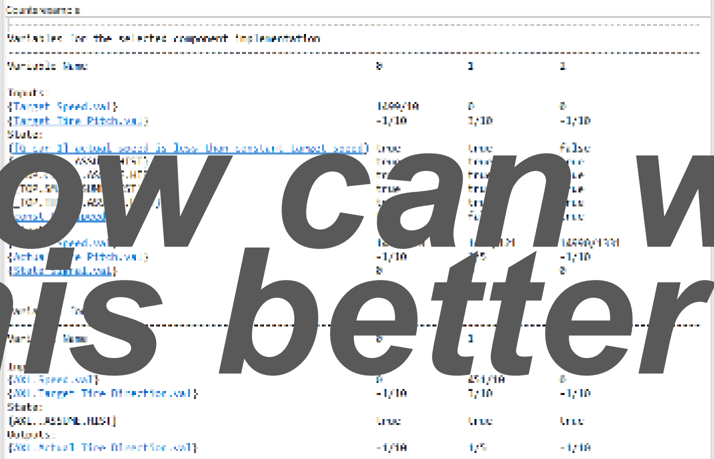
Making Sense of the Data

Creating Actionable Results

Lack of a Scalable Solution

Counter-Examples from Existing Tool Outputs are Difficult to Interpret

OSATE (Original)

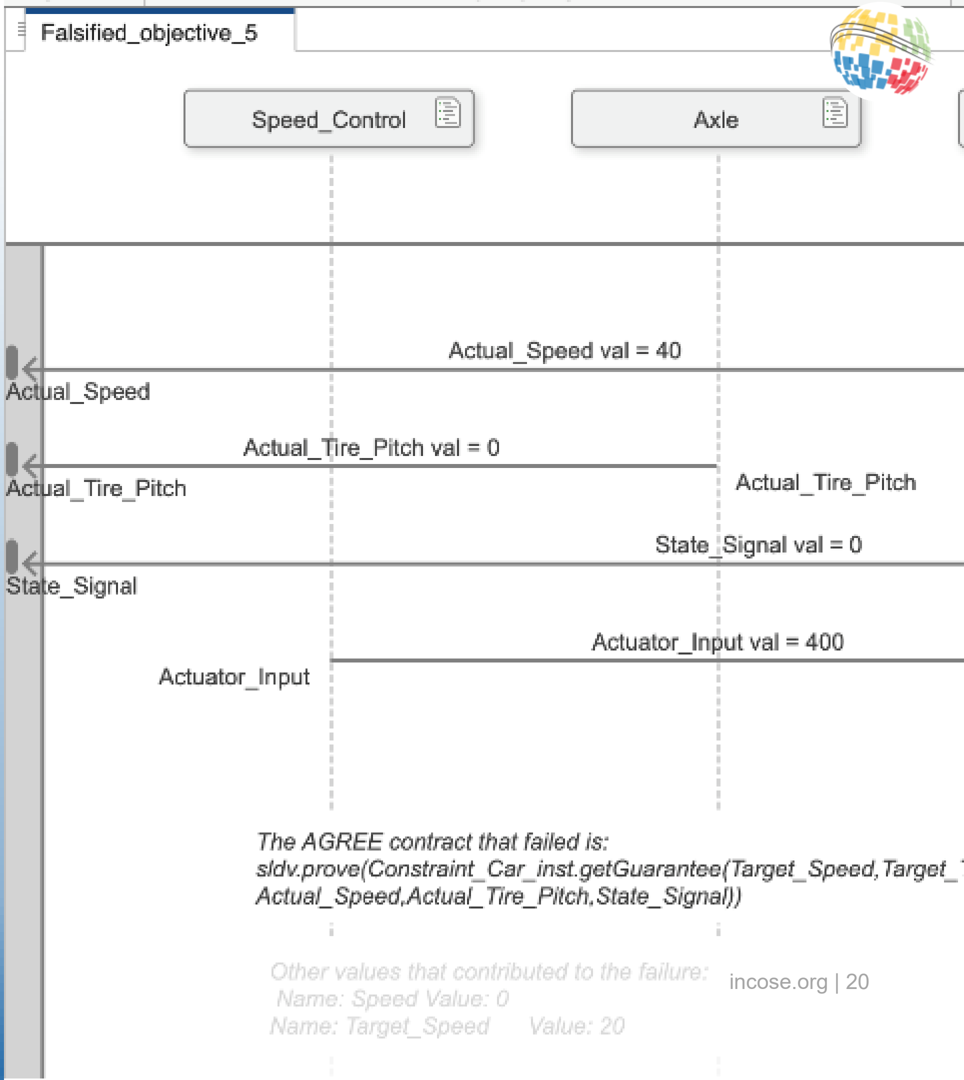


Simulink Design Verifier

How can we make this better?

Sequence Diagrams!

Sequence Diagrams provided the perfect medium for conveying human-readable Assume/Guarantee Counter-Examples



Where Do We Go from Here?

Key Takeaways and Next Steps

What We Did

The primary goal of this work was to make MBSE-based formal analysis **more accessible** to the systems engineering community.

- Demonstrated how to tag system components with formal behavioral contracts traced to system requirements
- Presented our approach for explainable counterexamples from the analysis results
- Applied AGREE-like compositional reasoning to a widely-used MBSE tool, System Composer
- Provided case studies demonstrating compositional reasoning and compared our results with semantically equivalent AADL/AGREE models
- Made our contribution available to the community through a MATLAB toolbox

Next Steps

- Scale Up Model Complexity
- Explore Hybrid Contract-Behavioral Models
- Use the Generated Sequence Diagrams for System Verification
- SysML v2.0 Support

Questions?

Josh Kahn
Principal Systems Engineering Strategist

joshkahn@mathworks.com
linkedin.com/in/josh-kahn-mbse

Contact the authors to request a copy of this MATLAB toolbox to give it a try yourself!