Formal Analysis of Software Architecture Models

AFRL Safe & Secure Systems & Software Symposium (S5)
June 3, 2009

Darren Cofer

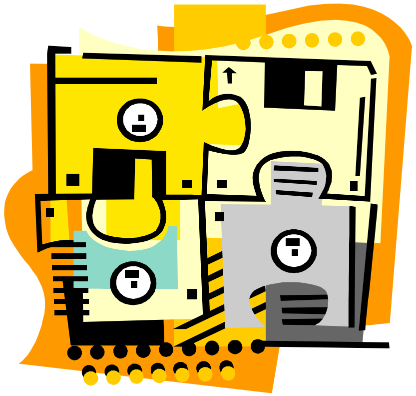
Rockwell Collins

Outline

- Vision
- Problem
- Current approaches
- Research directions

Vision: “Integrate, then Build”

- Build on success of model checking for software components
- Extend to system level via software architecture models
- Goal: Early detection/elimination of bugs
  - cheaper to fix in design vs. integration
- Hardware analogy…

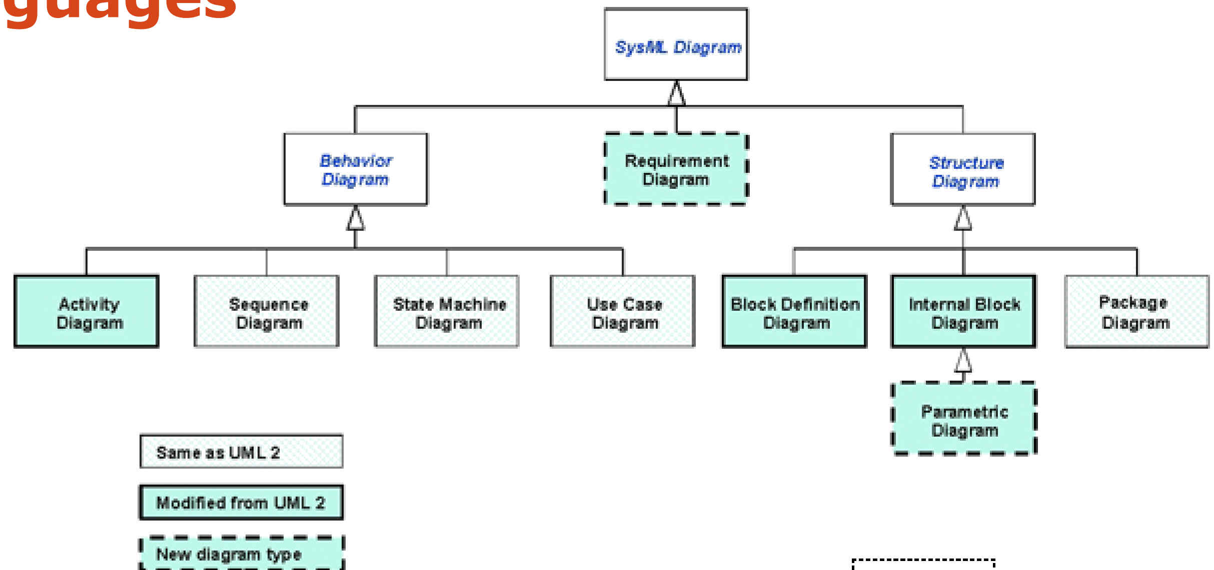


Software architecture model

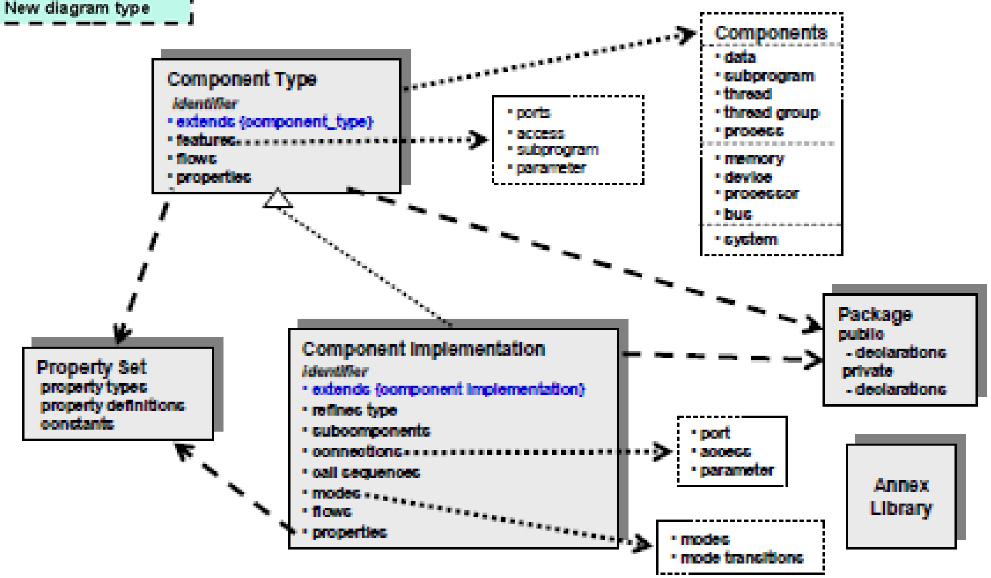
- Architecture defines interactions between components
  - Physical interconnect may be abstract or concrete (e.g., bus)
  - Includes fault-tolerance mechanisms such as redundancy
  - Actual component behavior abstracted or not specified
- Usually defines interactions with external environment
  - Sensors
  - Actuators
- May also specify behavior of execution platform
  - Scheduling
  - Software/hardware allocation
  - Synchrony/asynchrony

Modeling languages

- SysML



- AADL



Barriers to analysis of architecture models

- Complexity
  - Component models are already large
  - Naïve approaches won’t scale
- Asynchrony
  - Aircraft systems are typically distributed
  - Locally synchronous but globally asynchronous
  - Asynchronous interactions are more complex to analyze
    - interleaving, execution order, variable time delays
- Behavioral interface specification
  - Components interface to each other and architectural elements
    - power, bus, discretes, cabinet
  - ICDs typically specify signals, messages/formats, connectors
  - Need to specify component behaviors that characterize their interactions with the rest of the system

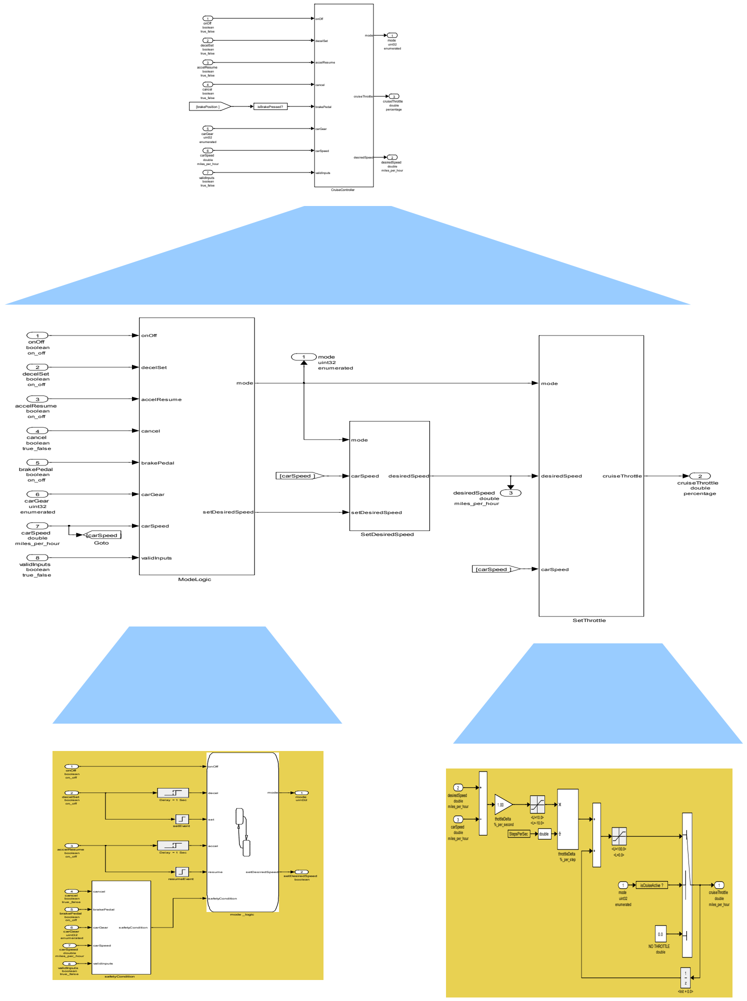
Problem

- To analyze system-level properties, we generally need to know about both the architecture and the components
  - How should we include component behaviors in a formal analysis of the system model?
- Flattening
  - Retain exact description of component behavior
  - Yields one big, hairy model of the whole system
- Abstraction
  - Replace component with simplified model
  - This is usually some type of state machine model

Current approaches

- AADL
  - Verimag: aadl2sync
    - flattening, quasi-synchronous
  - SPICES ITEA project: aadl2bip
    - abstraction via Behavior annex
  - TOPCASED => FIACRE => TINA
  - UIUC: MOMENT2-AADL => Maude
- SysML
  - state machines
  - activity diagrams => Petri nets
- others

A different approach…



Typical Model-Based Design

- Models are organized in a hierarchy several (many) levels deep
- Most of the complexity is in the leaf models
- Leaf models can often be verified through model checking

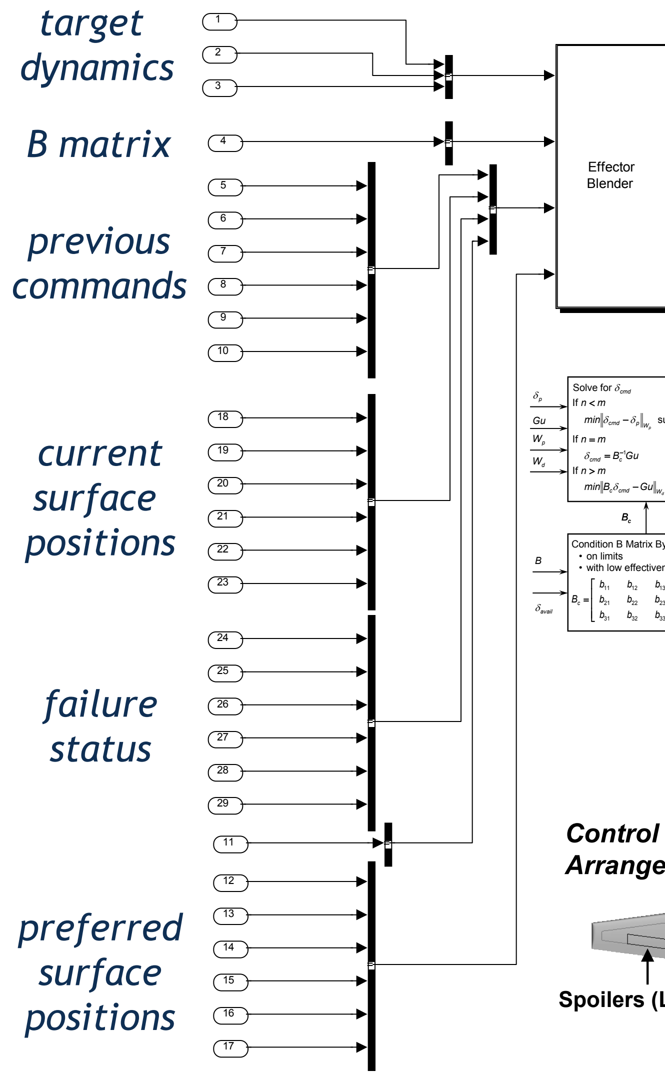
Composition of Subsystems

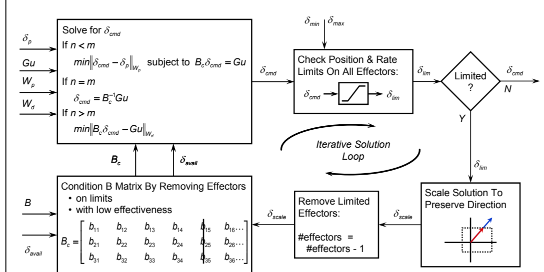
- Tends to be simple
- Assume/guarantee reasoning
- Well suited for theorem proving

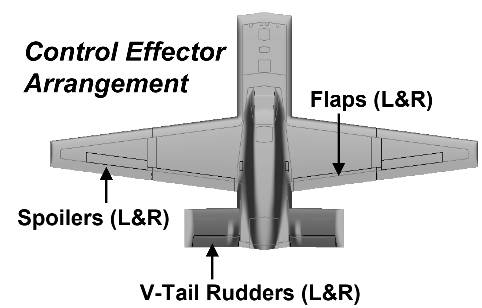
Effector Blender Model

- Large Complex Model
  - 166 Simulink subsystems
  - 2000+ basic Simulink blocks
  - Huge reachable state space
  - *This is bad*
- Completely Functional
  - No internal state
  - *This is good*

Can we use an approach based on *compositional reasoning* to handle the complexity of this system?



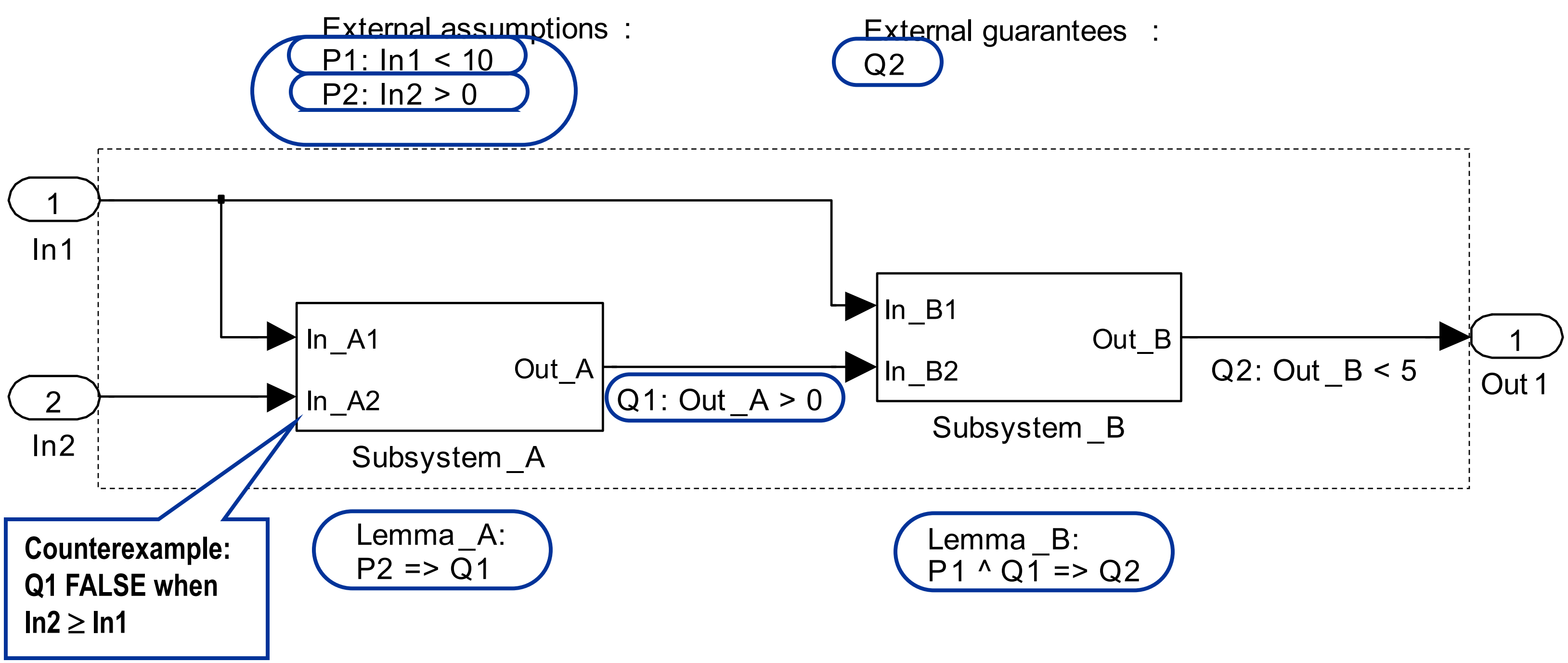




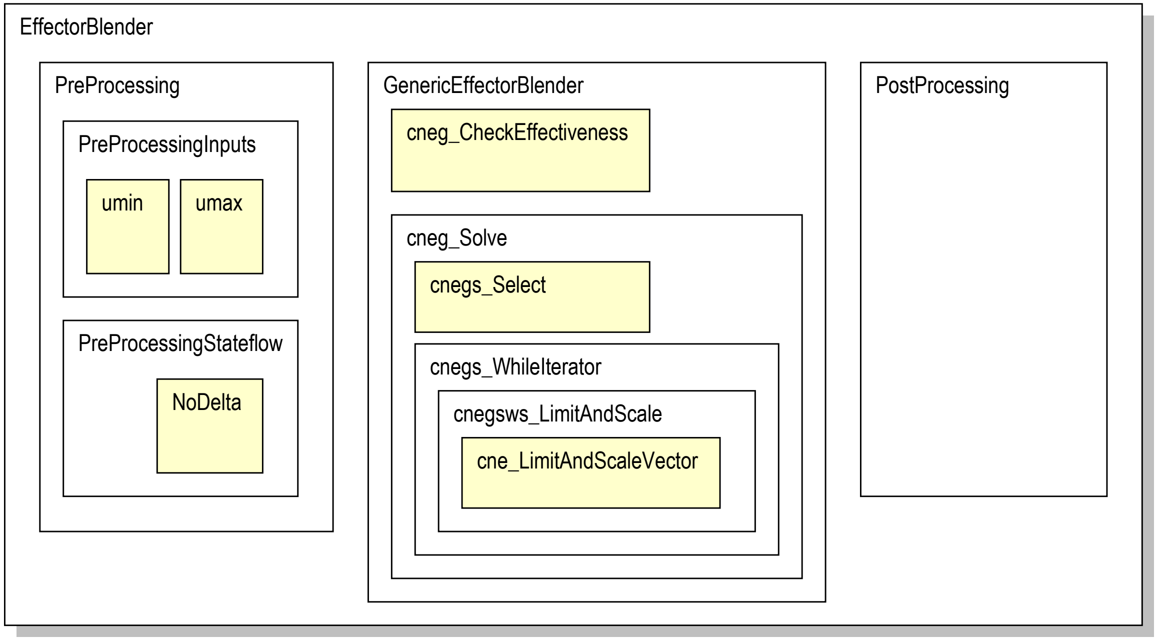
Compositional reasoning

Hybrid verification process

- Decompose system property into a set of lemmas corresponding to individual subsystems
- Proof = argument that lemmas imply truth of property + demonstration that lemmas are true for subsystems
- Lemmas are verified by model checker



CerTA FCS: Effector Blender analysis



Properties

- umin ≤ u ≤ umax
- If no failure, surface commands never exceed static limits
- If failure, surface commands stay within +/- current position
- Surface commands never exceed rate limits

- (dis-)Proof carried out iteratively, starting with top-level obligations imposed by properties, and propagating through the architecture
- Started with cne_LimitAndScaleVector subsytem, which performs command limiting function
- Lemmas developed and propagated outward until satisfied or counterexamples found

Formal analysis of architecture models

- Most current approaches are based on AADL Behavior Annex or SysML State Machines
- This is a useful abstraction for some classes of behaviors…
- …but it is not general and does not fit well with model-based development process for software components
- We want an approach that builds on the current success of model checking SW components
- Idea: Follow same approach as the compositional reasoning example
  - Property language to define assumptions/guarantees, requirements, environmental constraints
  - Decompose requirements and allocate to components and interactions
  - Verify properties at "leaf nodes" by model checking
  - May need to generate test cases to verify some properties
  - Reason about results at system level
  - May use a combination of theorem proving and model checking

Research directions

- Expand domain: architectural analysis
  - Build on successful analysis of SW at component level
  - Handle mixed synchronous/asynchronous systems
  - Tooling to automate lemma generation/propagation
  - Combination of theorem proving and model checking
- Property-based verification
  - Common specification language for properties, constraints, assumptions
  - Properties drive testing and analysis
  - Develop assurance arguments to combine evidence and demonstrate complete coverage
- Architectural patterns
  - Reusable design patterns that have *proven* properties
  - Ex: PALS (for async bounded delay networks), redundancy mgt
- Expand scope of analysis tools
  - Improvements in decision procedures allow analysis of larger classes of systems