Complexity-Reducing Design Patterns for Cyber-Physical Systems

DARPA META Project

AADL Standards Meeting
24-27 January 2011
Steven P. Miller
META is Part of the DARPA AVM Program

Shorten development times for complex defense systems [META]
- Raise level of abstraction in design of electromechanical systems
- Enable correct-by-construction designs through model-based verification
- Compose designs from component model library that characterizes the “seams”
- Rapid requirements trade-offs; optimize for complexity & adaptability, not SWaP

Shift product value chain toward high-value design activities [iFAB]
- Bitstream-configurable foundry-like manufacturing capability for defense systems
- Rapid switch-over between designs with minimal learning curve
- “Mass customization” across product variants and families

Democratize design [FANG]
- Crowd-sourcing infrastructure to enable open-source development of electromechanical systems [vehicleforge.mil]
- Series of prize-based Adaptive Make Challenges culminating a Ground Combat Vehicle prototype for evaluation against Army GCV Program of Record [FANG]
- Motivate a new generation of designers and manufacturing innovators [MENTOR]
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
In modern combat systems, the **software architecture** is the integration mechanism for electrical, mechanical, and software components.

Software development, including verification, poses the **single greatest risk** to program schedule and successful delivery.

- 40% of Commercial Aircraft Cost due to [cyber-physical] Systems
  - Don Winter, VP Boeing, CPS Transportation Workshop, November 2008
- SLOC Doubles Every Four Years in Commercial Aircraft
  - AVSI SAVI Summary Final Report, October 8, 2009
- Current Generation of Commercial Aircraft Software Exceeds 25M SLOC
  - AVSI SAVI Summary Final Report, October 8, 2009
- Cost of 27.5M SLOC of Commercial Aircraft Code = $10 Billion
  - AVSI SAVI Summary Final Report, October 8, 2009
- 30% of Automobile Cost Due to Electronics
  - US Council for Automotive Research
- 2006 Estimate of 34M SLOC for Future Combat Systems
  - The Army’s Future Combat System Program, CBO, April 2006
- Future Systems will Likely Exceed One Billion SLOC
  - Don Winter, VP, Boeing, Congressional Testimony, NITRD Program, July 31, 2008

Develop and demonstrate a new design flow based on **complexity-reducing design patterns** with formally guaranteed properties, and **automated formal verification** deeply embedded within the system design process.
Complexity-Reducing Design Patterns for Cyber Physical Systems

Objective

• Achieve dramatic reduction in the time required to design and verify complex, mixed-criticality cyber-physical systems

Key innovations

• Complexity-reducing system design patterns with formally guaranteed properties
• Architectural modeling and analysis to support virtual integration, composition, and verification of system-level properties
• Automated formal verification deeply embedded in the system design process

Impact

• Dramatic schedule efficiencies
• Correct by construction eliminates rework cycles
• Integrated verification eliminates rework & retest ➞ direct to foundry

Team

• Rockwell Collins ATC
• University of Illinois at U-C
• University of Minnesota
• WW Technology Group

Technology Transition

• Focus on open standard modeling languages
Complexity-Reducing Architectural Design Patterns

• Design pattern = model transformation
  – \( p : M \rightarrow M \) (partial function)

• Reuse of verification is key
  – Not software reuse
  – Guaranteed behaviors

• Reduce/manage system complexity
  – Separation of concerns
  – System logic vs. application logic
  – Process complexity vs. design complexity

• Encapsulate & standardize good solutions
  – Raise level of abstraction
  – “Take the soldering iron out of the hands of systems engineers”
### Task 1 - Design Problems

- Evaluate design problems and possible solutions

- Data sources
  - SAVI ROI report
  - RC SCR database
  - Fault-tolerance literature

- Patterns
  - PALS
  - Replication
  - Leader Selection
  - Active-Standby
  - Simplex

<table>
<thead>
<tr>
<th>Design Problem</th>
<th>Description</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Asynchronous Computation</td>
<td>Race conditions lead to inconsistent state information across asynchronous nodes.</td>
<td>Implement logical synchrony to simplify agreement on global state.</td>
</tr>
<tr>
<td>2 Unreliable Computing Platform</td>
<td>Incorrect designs for fault tolerance result in system failures.</td>
<td>Use of verified architectural design patterns for fault tolerant design.</td>
</tr>
<tr>
<td>3 Unreliable Data Sources</td>
<td>Incorrect designs for voting or selection of redundant sensors lead to use of invalid inputs.</td>
<td>Use of verified architectural design patterns for source selection and voting.</td>
</tr>
<tr>
<td>4 Unintended Interaction/Interference</td>
<td>Failure or unexpected behavior of a component causes the failure of a unrelated component.</td>
<td>Use of verified architectures that prevent interference. Use of assume/guarantee contracts.</td>
</tr>
<tr>
<td>5 Resource Contention</td>
<td>Incorrect use of shared resources leads to inconsistent global state, corrupted resource, or denial of service.</td>
<td>Use of verified architectural design patterns for concurrent access to shared resources.</td>
</tr>
<tr>
<td>6 Platform Hardware Dependencies</td>
<td>Dependencies on low-level hardware or platform behavior make it prohibitively expensive to port systems to new platforms.</td>
<td>Use of layered virtual interfaces based on formally specified assume/guarantee contracts. Automated generation of each level of design refinement.</td>
</tr>
<tr>
<td>7 SW-HW Task Allocation</td>
<td>Inflexible allocation of functions to hardware or software makes it difficult to optimize system performance/cost.</td>
<td>Auto-generation of correct-by-construction implementations from system specification.</td>
</tr>
<tr>
<td>8 Untrusted/Unreliable Components</td>
<td>Use of complex components that provide highly desirable performance are prohibitively expensive to verify.</td>
<td>Use of formally verified simplex design pattern that monitors boundary conditions and switches to a simpler reliable controller.</td>
</tr>
<tr>
<td>9 Requirements Errors</td>
<td>Missing or incorrect requirements are not discovered until system integration resulting in extensive rework.</td>
<td>Use executable system models and compositional verification of to enable early identification of requirements errors.</td>
</tr>
<tr>
<td>10 Unspecified Dependencies</td>
<td>Missing or incorrect dependencies of components on other components are not discovered until system integration.</td>
<td>Use of formal assume/guarantee contracts and compositional verification of to enable early identification of unspecified dependencies.</td>
</tr>
</tbody>
</table>
Task 2 - Design Flow

- Encapsulate Good Solutions
- Reduce System Complexity
- Pre (Formal) Verification
- Standardize System Design
- Reuse of Verification
- Compositional Reasoning
- Correct-By-Construction
- Avoid Redesign
Task 3 - Initial Design Patterns

- **PALS – Physically Asynchronous/Logically Synchronous**
  - Design and verify an asynchronous system as a synchronous system
  - Orders of magnitude reduction in temporal complexity
  - Eliminate deadlock and race conditions

- **Replication**
  - Replicates components, ports, and connections
  - Foundation for more complex fault-tolerance patterns

- **Leader Selection**
  - Common problem of selecting the active node in a distributed design
  - Formally (pre) verified for N nodes

- **Active Standby**
  - Common fault-tolerant design for hardware failures

- **Simplex**
  - Common fault-tolerant design for software failures
Task 3 - PALS Pattern

- **Goal:** Host synchronous designs “transparently” on distributed asynchronous platforms

- **Assumptions**
  - Access to global time
  - Monotonic local clocks
  - Bounded computation and communication times
  - Fail silent nodes

- **Guarantees**
  - All nodes see the same input values at the logical step boundaries
Task 4 – Example Architectural Models

Initial System

Active Standby Pattern

Replicate
Leader Selection
PALS

Replicate
Final System

Avionics System

Flight Control

Flight Guidance

Pattern Application

System Hierarchy

© Copyright 2010 Rockwell Collins, Inc. All rights reserved.
Task 5 - Pattern Verification

- **Leader Selection pattern**
  - Generic for \( n \) nodes
  - Synchronous (requires PALS)
  - Used to implement Active-Standby

- **Assume**
  - Nodes connected in model (can exchange messages)
  - Synchronous communication (dramatic state-space reduction)

- **Guarantee (for \( n \leq 8 \))**
  - All non-failed nodes always agree on a leader.
  - If a node is leader and has not failed, it is still the leader in the next step.
  - If any non-failed nodes exist, then in the next step there is a leader.
  - No failed node will be the leader in the next step.

- **Verified using NuSMV model checker**
  - Working on parametric proof for arbitrary number of nodes
Task 6 – Tools for Pattern Specification & Application

- WW Technology Group
- EDICT Tool Suite
- Pattern Specification
- Pattern Library
- Checking Pattern Constraints
- Pattern Application
- Generation of Revised AADL Model
Task 7 – Verify System Properties

- Compositional reasoning
  - Component and pattern specifications
  - Assume (environmental invariants)
  - Guarantee (properties/requirements proved)

- AFRL CerTA FCS Effector blender example
  - Proof carried out iteratively, starting with top-level obligations imposed by properties, and propagating through the architecture
  - Lemmas developed, model checked, and propagated outward until satisfied or counterexamples found
FAA Requirements Engineering Management Handbook

- Two Year Study for the FAA
- Recommended Practices
- Examples
- Based on SCR and CoRE
- Elements of SpecTRML
- Written with AADL in Mind
SysML, AADL, and MARTE

- **SysML**
  - Systems Modeling Language
  - Applicable Across Most of the Systems Engineering Domain
  - OMG Standard
  - Widely Known with Improving Commercial Tool Support
  - Lacks Specificity and Rigor for Embedded Systems

- **UML**

- **MARTE**
  - Modeling & Analysis of Real-Time Embedded Systems
  - Tailored to Real-Time Embedded Systems Domain
  - UML2 Profile
  - Best Known in France and Europe
  - Few Existing Tools with Uncertain Future Support

- **AADL**
  - Architectural Analysis and Design Language
  - Tailored to Real-Time Embedded Systems Domain
  - SAE Standard (AS5506)
  - Active Research Community, Particularly in Europe
  - Tools are Primarily Open Source with Uncertain Future Support

Intended Domain

Physical World

Real Time Embedded Systems

Software

Modeling

Intended Use

Analysis
Summary

- **Complexity Reducing Patterns for Cyber-physical Systems**

- **Objective**
  - Achieve *dramatic reduction* in the time required to design and verify complex, mixed-criticality *cyber-physical* systems

- **Key Innovations**
  - *Complexity-reducing system design patterns* with formally guaranteed properties
  - *Architectural modeling and analysis* to support virtual integration, composition, and verification of system-level properties
  - *Automated formal verification deeply embedded* in the system design process

- **Integrate ... Then Build**