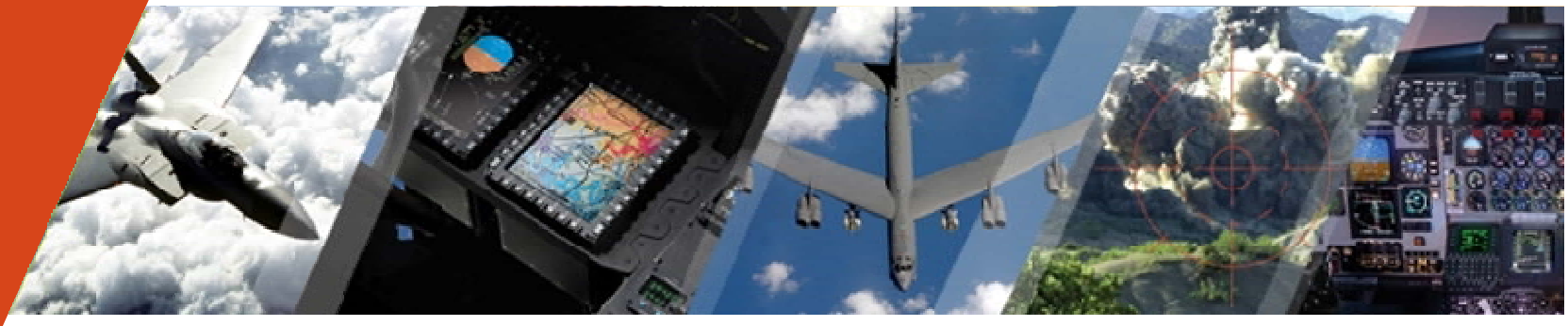
Complexity-Reducing Design Patterns for Cyber-Physical Systems

DARPA TTO - META

14-16 June 2011

Darren Cofer



Rockwell Collins

META is part of the DARPA AVM program

APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

Adaptive Vehicle Make vision

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]

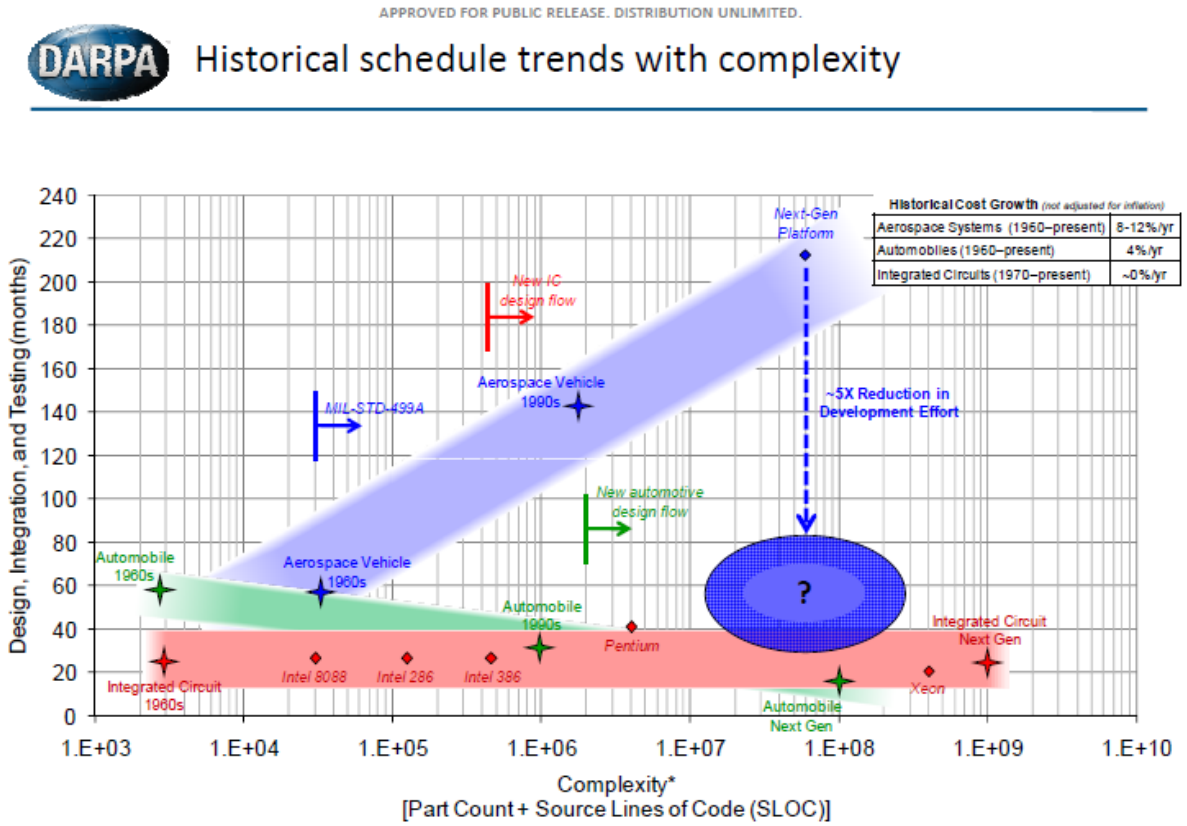
- Crowd-sourcing infrastructure to enable open-source development of electromechanical systems [vehicleforge.mil]
- Series of prize-based Adaptive Make Challenges culminating a Ground Combat Vehicle prototype for evaluation against Army GCV Program of Record [FANG]
- Motivate a new generation of designers and manufacturing innovators [MENTOR]

33

Rockwell Collins

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



Team

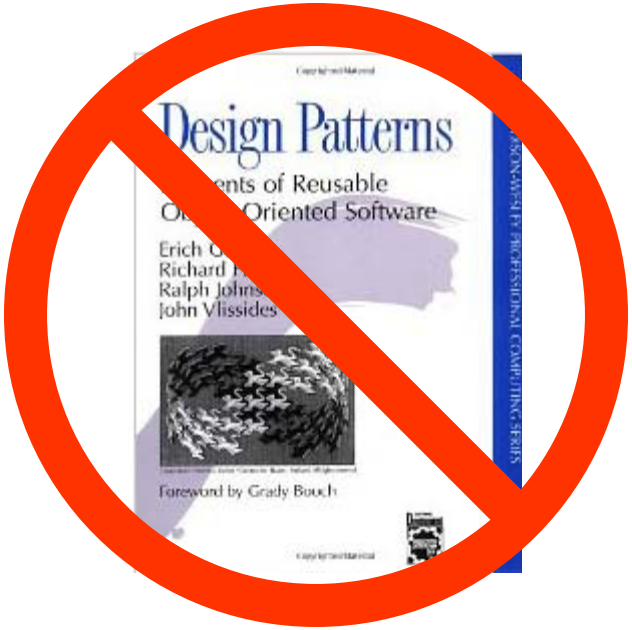
- Rockwell Collins / Advanced Technology Center
  - Darren Cofer, Steven Miller, Andrew Gacek
  - System modeling & analysis, tooling, integration
- UIUC
  - Lui Sha
  - Design pattern development
- University of MN
  - Michael Whalen
  - Pattern verification, compositional analysis
- WWTG
  - Chris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

Rockwell Collins

Topics

- Review: What is a design pattern?
- Key insights
- Results
  - Design flow and tools
  - PALS: vertical contract
  - Structural property checking
  - Contract between patterns
- Next steps

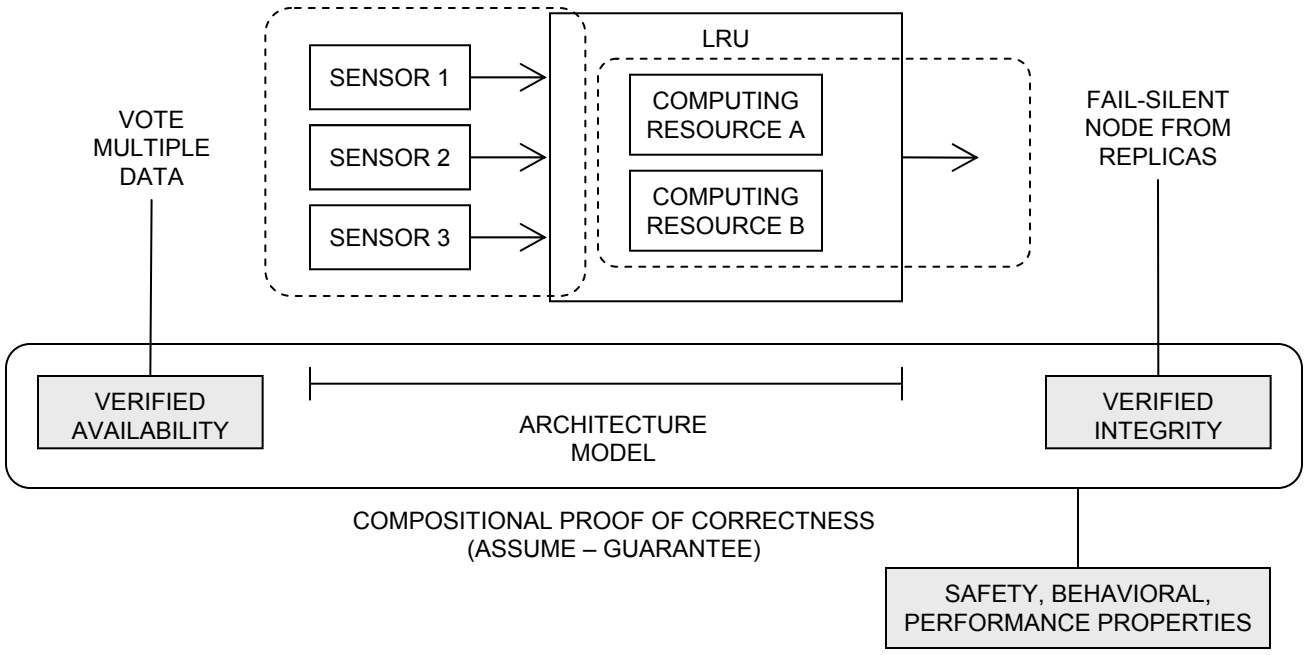
Complexity-Reducing Architectural Design Patterns



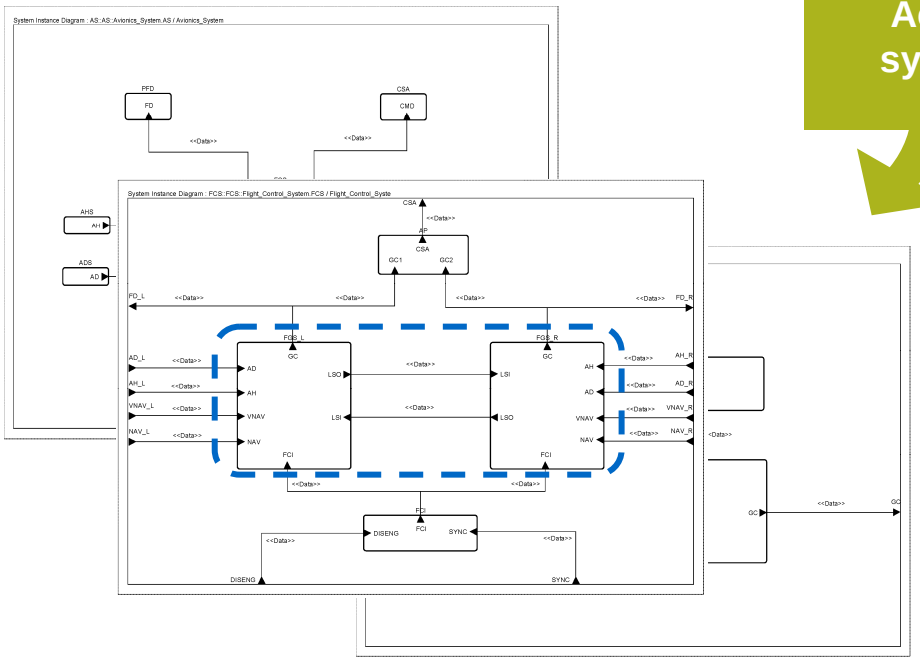
- Design pattern = model transformation
  - $p : \mathcal{M} \rightarrow \mathcal{M}$ (partial function)
  - Applied to system models
- Verification reuse is key
  - Not software reuse in OO style
  - Patterns (and components) provide guaranteed behavior
  - Formal verification effort amortized over many system designs
- Reduce/manage system complexity
  - Separation of concerns
  - System logic vs. application logic
  - Compositional reasoning exploits system hierarchy
- Encapsulate & standardize good solutions
  - Raise level of abstraction
  - Codify best practices

Vision

- System design & verification through pattern application



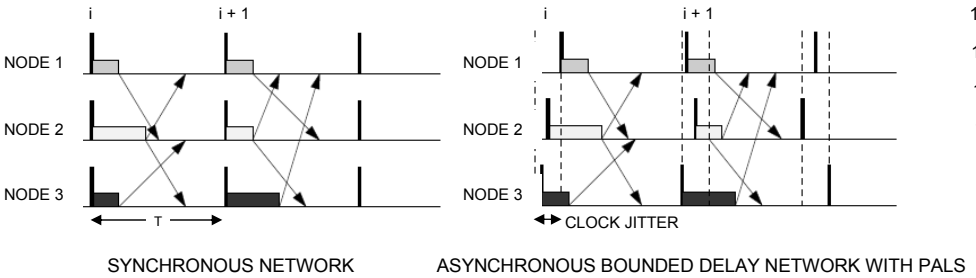
Design patterns attack system complexity through automated model transformations



Active-Standby pattern allows system developers to work at a higher level of abstraction

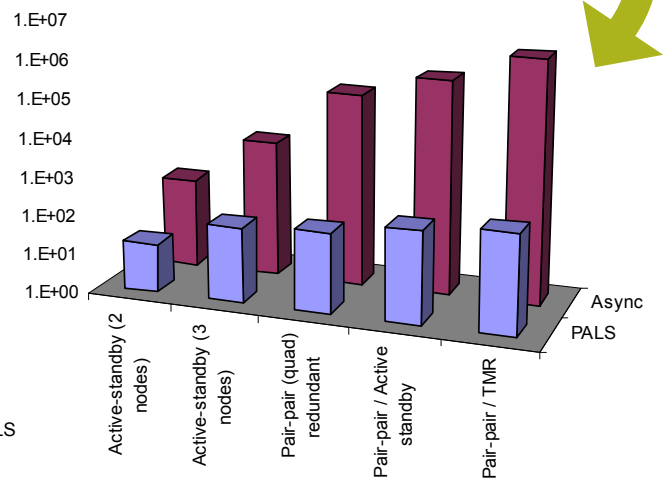
"Use of formally verified Active/Standby design cut development time by 1/3 and saved hundreds of hours of on-aircraft test time." — RC Commercial Systems

PALS pattern achieves >3 orders of magnitude reduction in state space and verification complexity



SYNCHRONOUS NETWORK

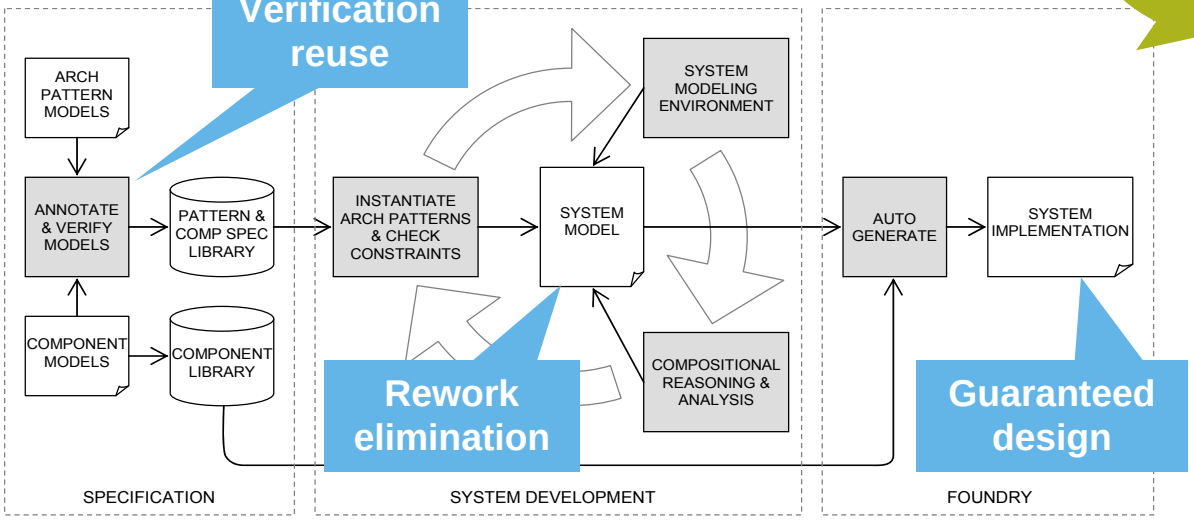
ASYNCHRONOUS BOUNDED DELAY NETWORK WITH PALS

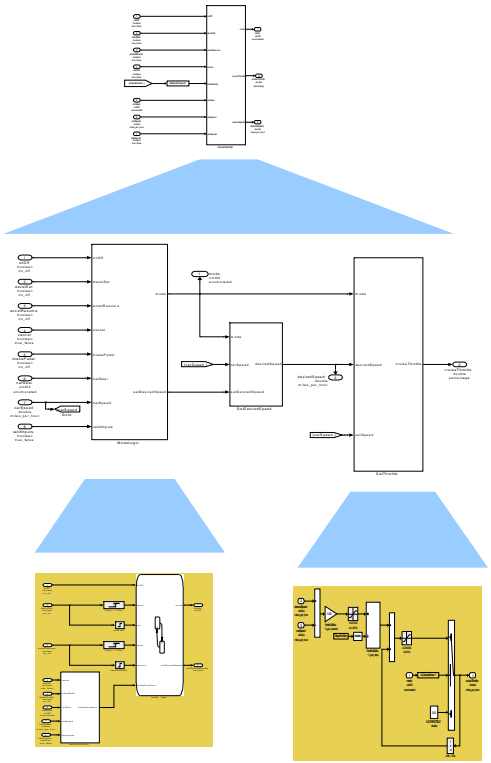


Powerful system synthesis tools based on pre-verified design patterns achieve dramatic reduction in rework and testing effort

Verification effort amortized over many designs as basis for correct-by-construction system design

Compositional verification exploits natural system hierarchy through formal assume-guarantee reasoning





ASSUMPTIONS GUARANTEES

"Rework cost is up to 60% of total development cost for large, complex systems." — AVSI SAVI

Initial design patterns

- PALS
  - Just enough synchronization
- Replication
  - Foundation for more complex fault-tolerance patterns
- Leader Selection
  - Set of nodes agree on leader

- In the works…
  - Voter
  - Simplex
  - PALS Whiteboard

Assumptions:
PALS Causality Constraint
PALS Period Constraint
Guarantees:
Period equals PALS Period
Synchronous comm

Assumptions:
Not co-located
Less than N faults
Guarantees:
One operational

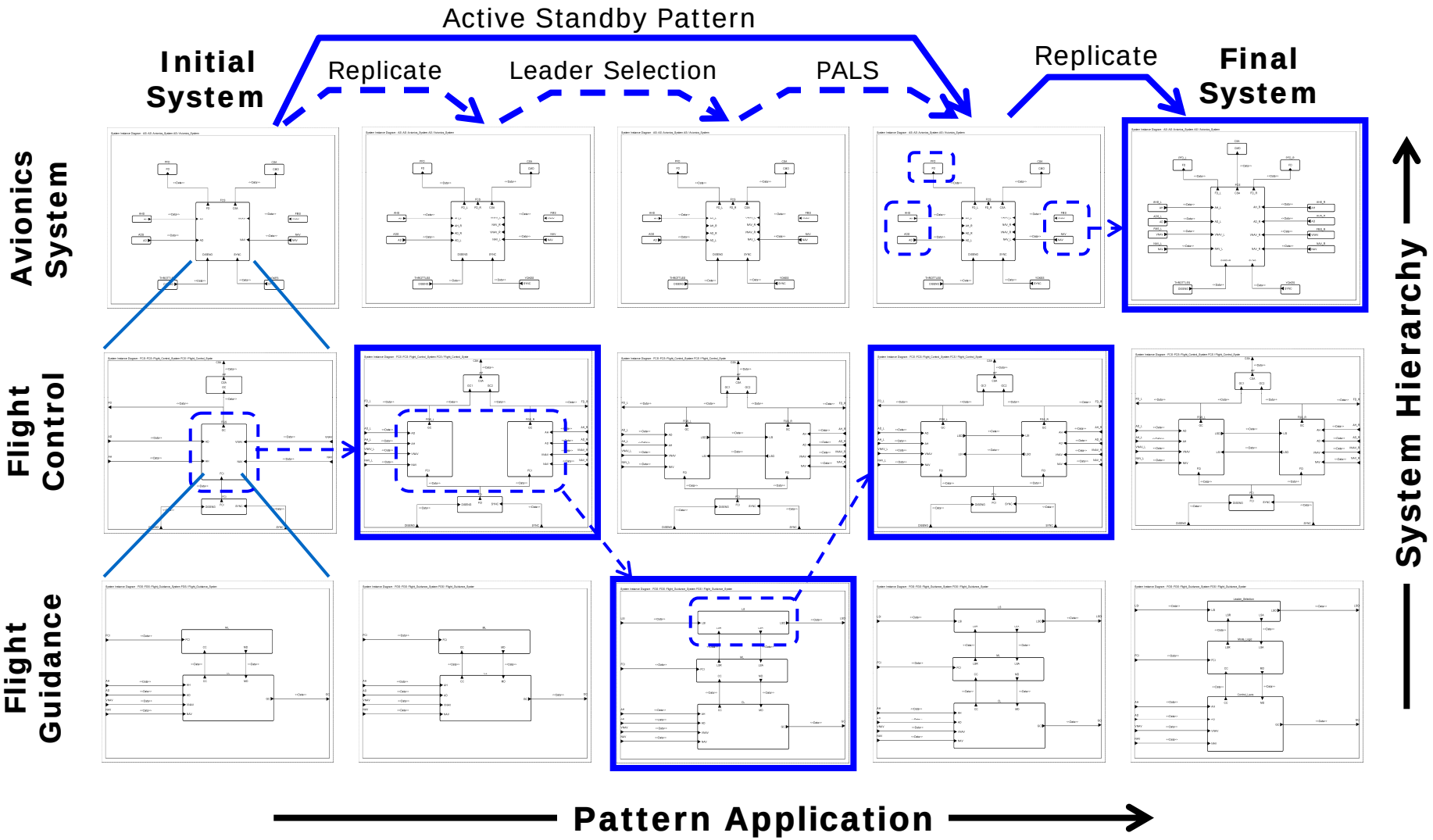
Assumptions:
One operational
Synchronous comm
Guarantees:
Leader exists
Leader non-failed
Non-failed nodes agree
Non-failed leader unchanged

Pattern design

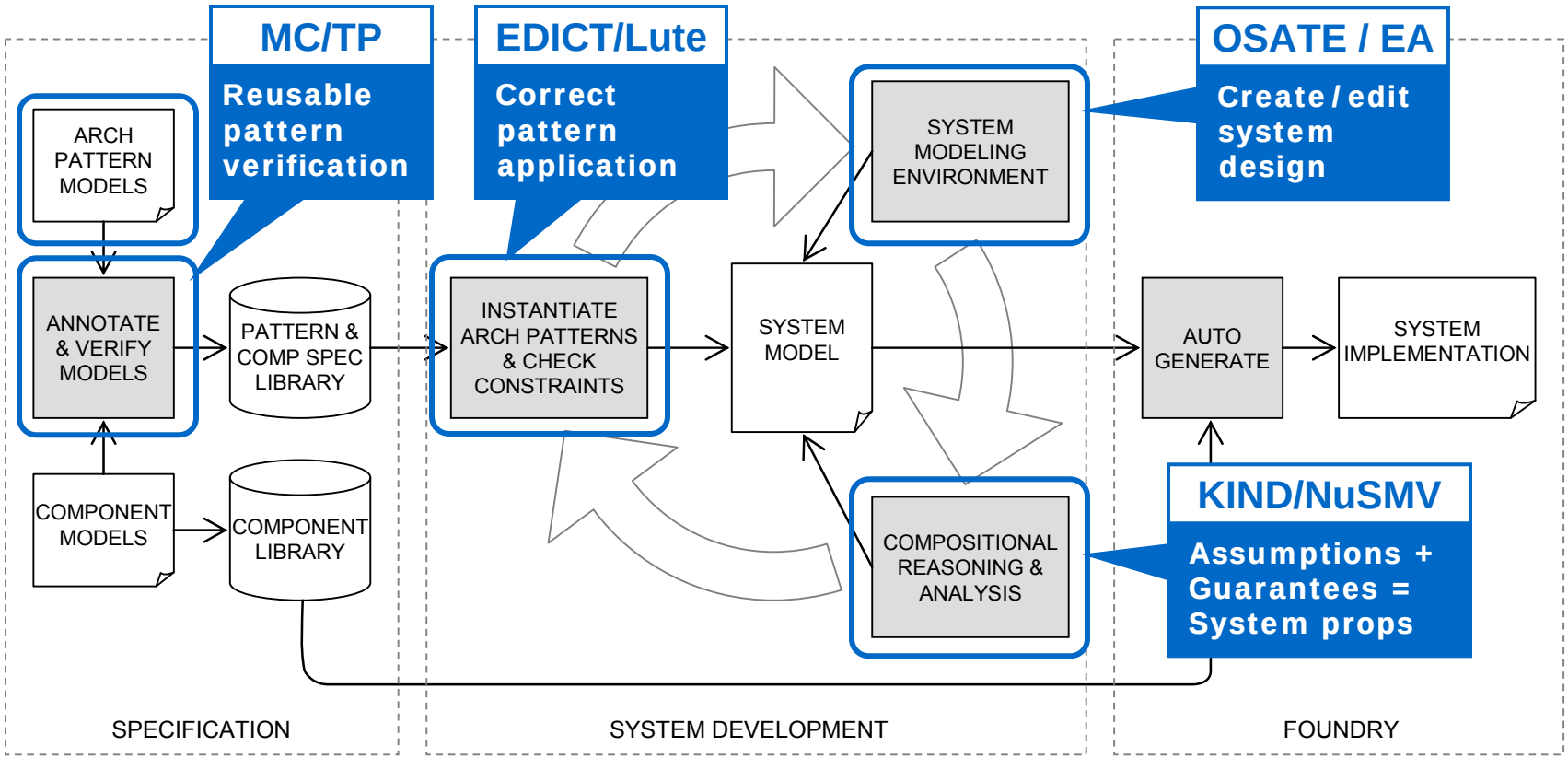
- Build patterns from fundamental operations
  - Replicate component
  - Remove component
  - Rename component
  - Insert component
  - Insert data specification
  - Replicate feature
  - Rename feature
  - Create feature
  - Remove feature
  - Create connection
  - Remove connection
  - Insert property set
  - Assign property
- Build larger patterns from smaller patterns
  - Active-Standby = Replication + Leader Selection + PALS
- Pattern can include structural constraints on models for instantiation
  - Ex: only apply PALS to leaf nodes
- Guaranteed behaviors of patterns are verified separately
  - Added to patterns as new AADL properties

Rockwell Collins

System Design Through Pattern Application



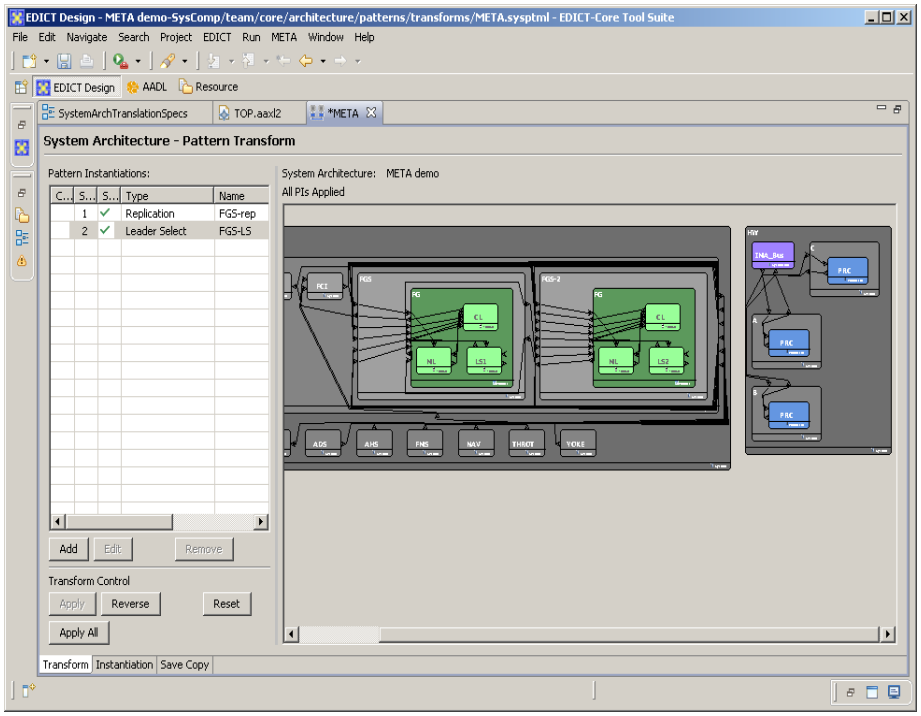
Design flow

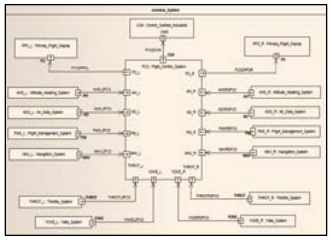


Tool architecture

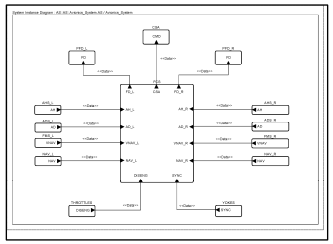
- Eclipse
  - Tool integration using plug-ins
  - Eclipse Modeling Framework
  - Builds on existing AADL tools

- Plug-ins
  - OSATE: AADL editing (SEI)
  - EDICT: Pattern instantiation (WWTG)
  - META: Import SysML from EA (RC)
  - META: Structural property checking by Lute (RC)
  - META: Assume-guarantee verification (UMN) – TBD

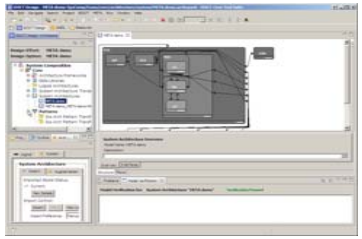




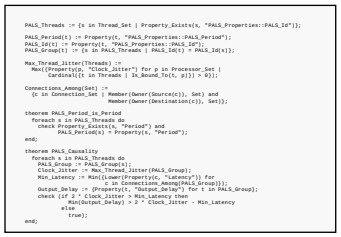
SysML



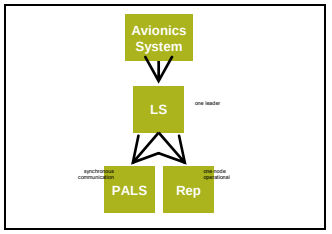
AADL



Patterns



Structural checks



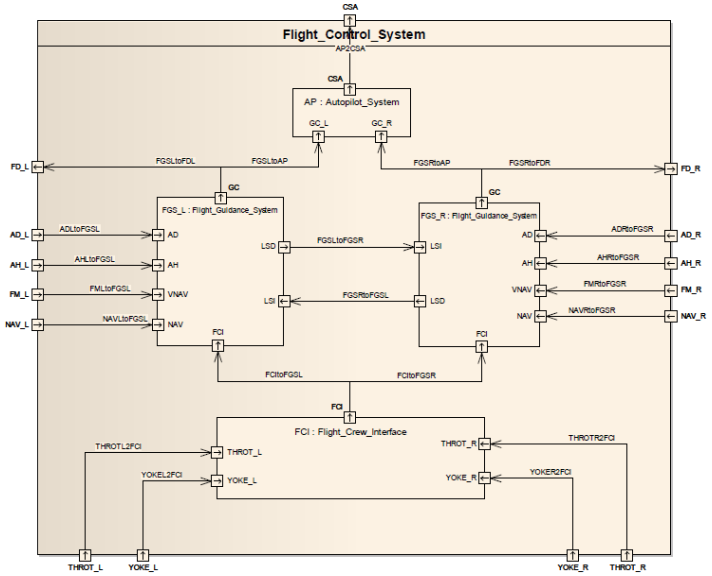
System verification

System architecture model

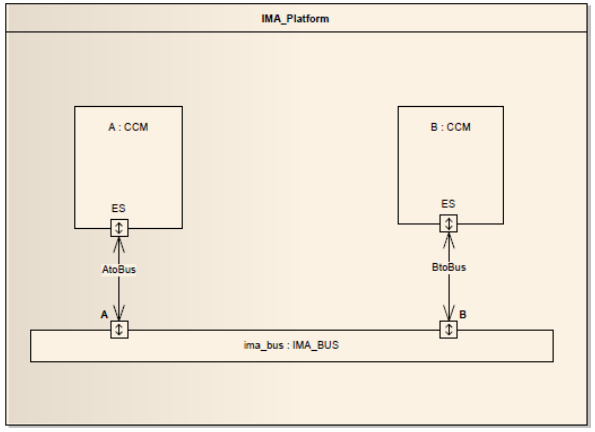
- Software + HW platform
  - Process, thread, processors, bus
- PALS vertical contract
  - PALS timing constraints on platform
  - Check AADL structural properties
- Guarantees
  - Sync logic executes at `PALS_Period`
  - `Synchronous_Communication => “One_Step_Delay”`
- Assumptions (about platform)
  - Causality constraint:

    Min(Output time) ≥ 2ε – μmin
  - PALS period constraint:

    Max(Output time) ≤ T - μmax - 2ε

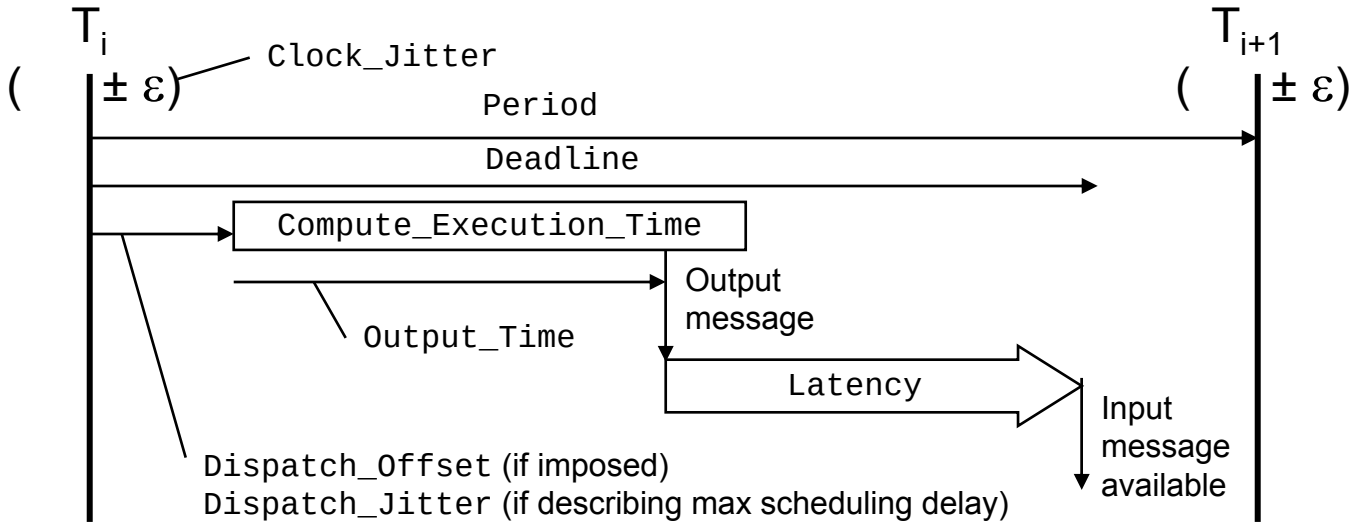


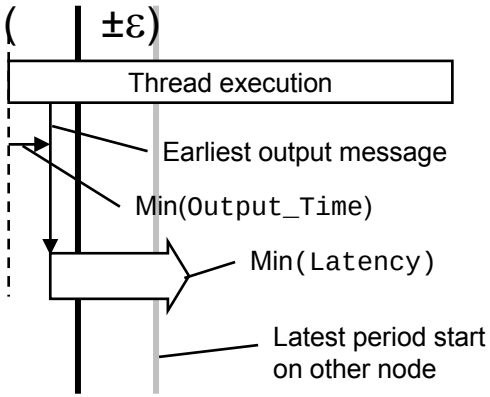
Software



Platform

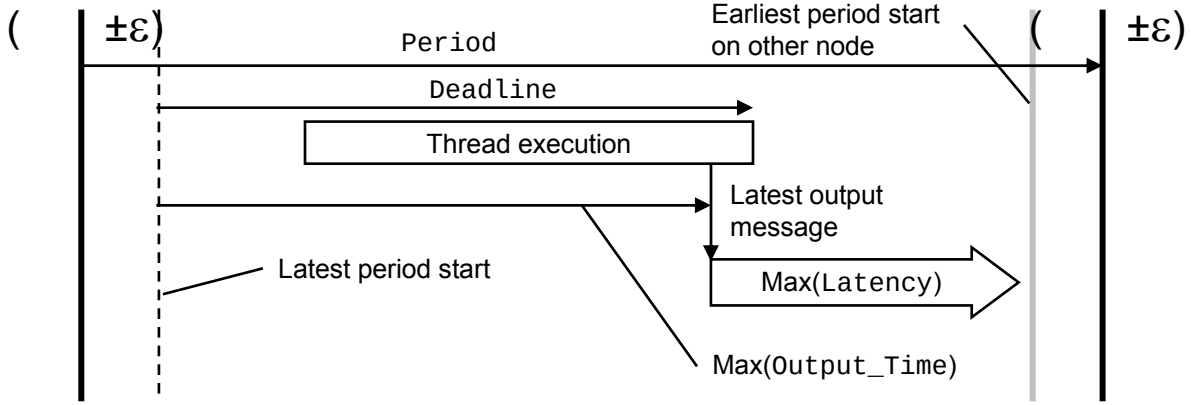
PALS assumptions in AADL model





Causality Constraint

Messages don't arrive too soon



PALS Period Constraint

Messages don't arrive too late

Structural property checks

- 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;
```

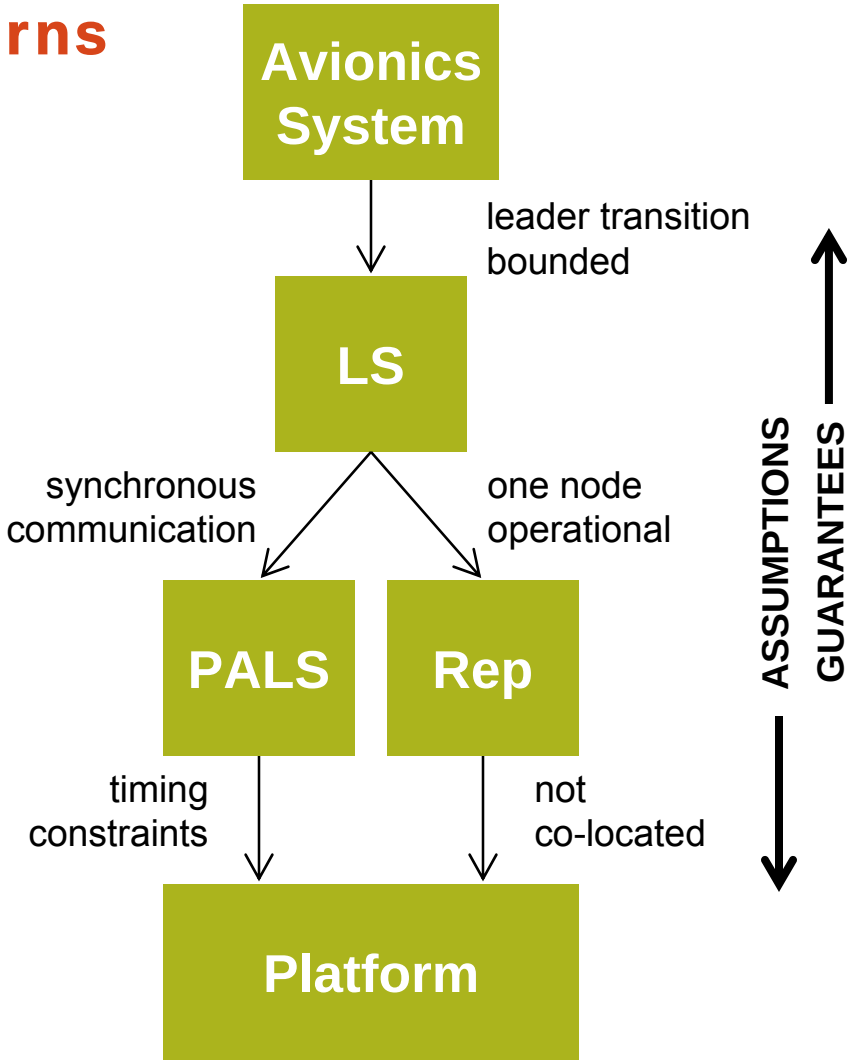
Contracts between patterns

- 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
  - System-level fault assumptions

Principled mechanism for "passing the buck"



Categories of properties

- Behavioral
  - Pattern and component interactions
  - Specified in PSL, verified by model checking
  - *Failed node will not be leader in next step*
    `G(!device_ok[j] -> X(leader[i] != j)) ;`
- Structural
  - Properties of the transformed model
  - Pattern assumptions, post-conditions
  - Specified and checked using Lute
  - *PALS period constraint*
    `Deadline < PALS_Period - Max_Latency - 2*Clock_Jitter`
- Resource allocation
  - RT schedulability, memory allocation, bandwidth allocation
  - ASIIST tool (UIUC/RC)
  - *Threads can be scheduled to meet their deadlines*

Rockwell Collins

Next steps

- Compositional techniques for system verification
  - Assume-Guarantee ledger
- Continue development of pattern instantiation tool
  - Implement additional patterns (Voting, Simplex)

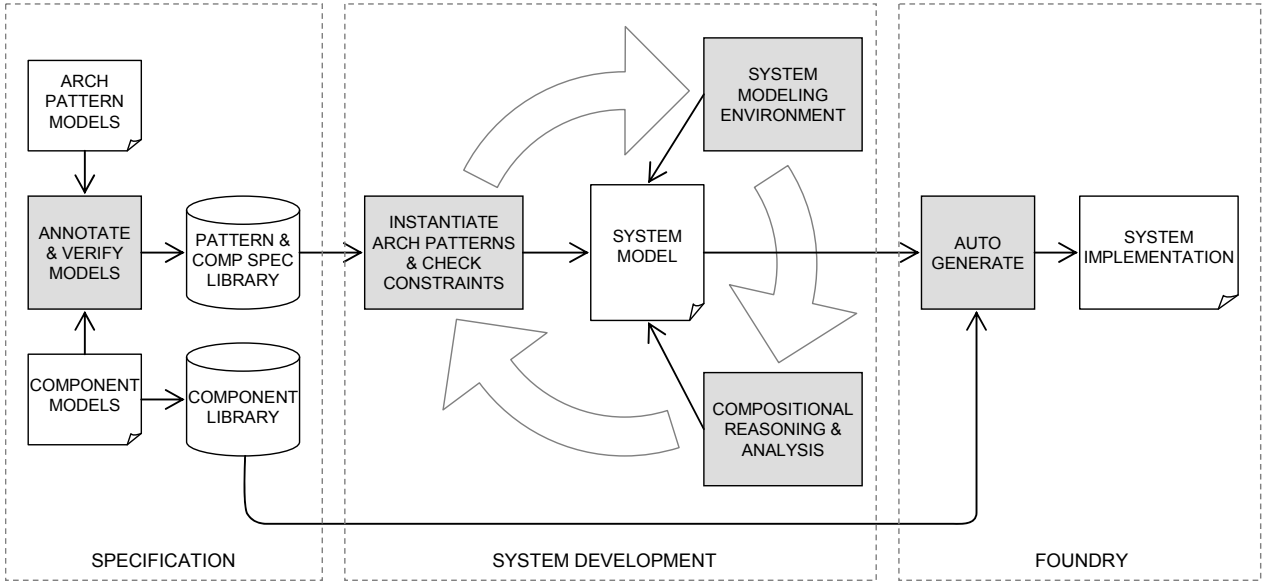
Complexity-Reducing Design Patterns for Cyber Physical Systems

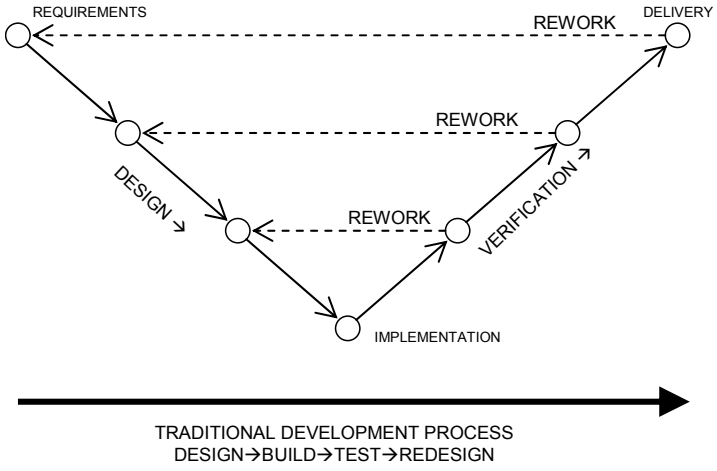
Objective

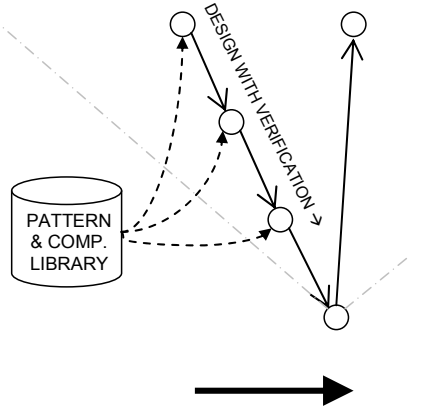
- Achieve dramatic reduction in the time required to design and verify complex, mixed-criticality cyber-physical systems

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 itself







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 U-C
- University of Minnesota
- WW Technology Group

Technology Transition

- Focus on open standard modeling languages