What is Composition? Why is it Hard?

Mike Whalen[1,2], Dan Bliek[2], Karl Hoech[2]

[1] UMSEC, University of Minnesota
[2] Rockwell Collins

Material includes insights gained from Gerrit Muller, John Shaw, Lee Pike, David Hardin, Bashar Nuseibeh, and others.

Abstract # 18869

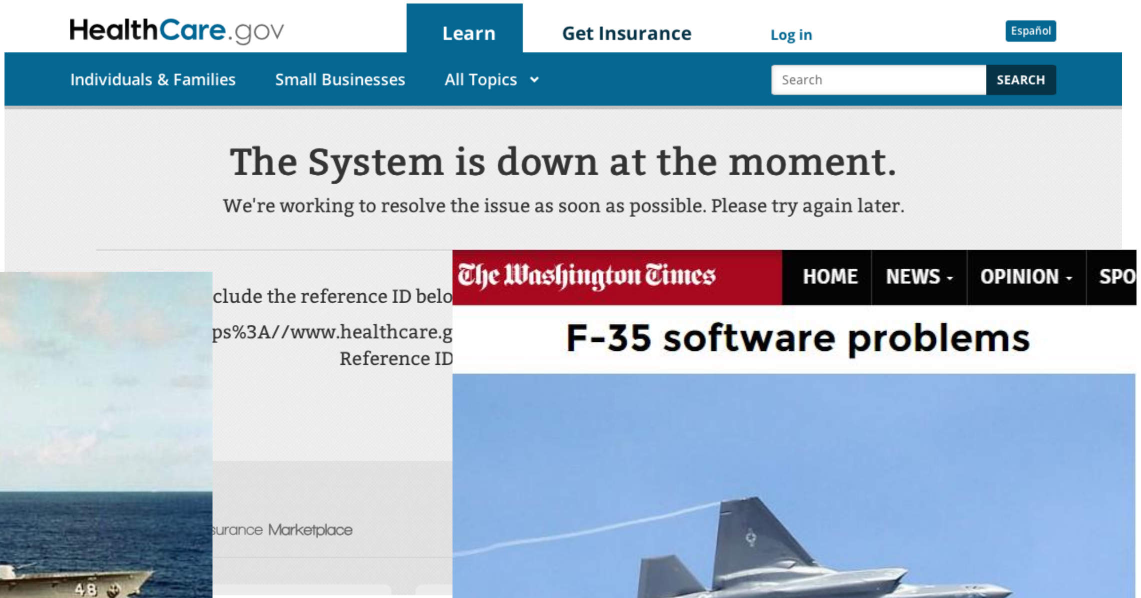
This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Rockwell Collins
Building trust every day

State of the Practice

- Systems (esp. those with significant software) do not to integrate easily or reliably.

- This will continue with systems of systems



HealthCare.gov

The System is down at the moment.

We're working to resolve the issue as soon as possible. Please try again later.

clude the reference ID belo

ps%3A//www.healthcare.g

Reference ID

Software glitches leave Navy Smart Ship dead in the water

F-35 software problems

Integration and Composition

- Organizations tend to have a “Parts” focus.
  - For “parts”, well-established:
    - Certification procedures
    - Engineering workflow
    - Documentation procedures
    - Artifact management (SCM, CAD, ERP, Docs)
    - …

- Integration is approached as static activity “wiring together” different subsystems
  - Focus is on composability of simple interface types
  - Behavior of the composed system is secondary, if considered at all
  - Non-functional attributes (especially performance) may be considered, but not from a reasonable basis

- **Little understanding or focus on key performance parameters**

Typical Order of Integration Problems

1. The (sub)system does not build.
2. The (sub)system does not function.
3. Interface errors.
4. The (sub)system is too slow.
5. Problems with the main performance parameter, such as image quality.
6. The (sub)system is not reliable.

List from: *Gerrit Muller: Why is System Integration Understood So Poorly?*
2016 Kongsberg System Engineering Event

A Change in Focus

- **Composition** is creating new or improved **capability** from a set of parts
  - Not "wiring", but behavior
  - Dynamic view of the system
  - Must require less effort and time than constructing from scratch

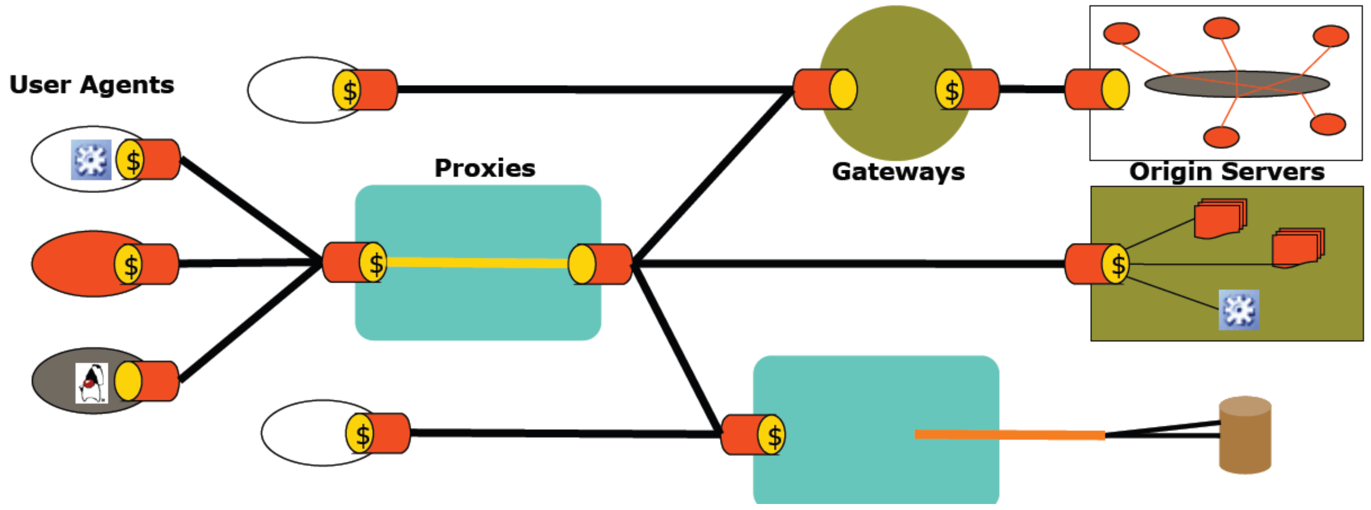
- Integration is a continuous, repeatable **process** not an event
  - It occurs before systems are built
  - Models are continuously updated with data
  - **Fail Early** rather than **Fail Late**

- Analysis of composition should have **analytic rigor**
  - Simulations are expensive and incomplete
  - Often simulations require "nearly complete" software
  - If possible, we would like **proof** or **optimal result**

- "Parts" should be easily reconfigurable to meet new nonfunctional goals
  - "Integration Architectures" vs. "Functional Architectures" (c.f. Evan's talk)
    - **Functional Architecture:** What we want the system to do
    - **Deployment Architecture:** How the pieces are assembled

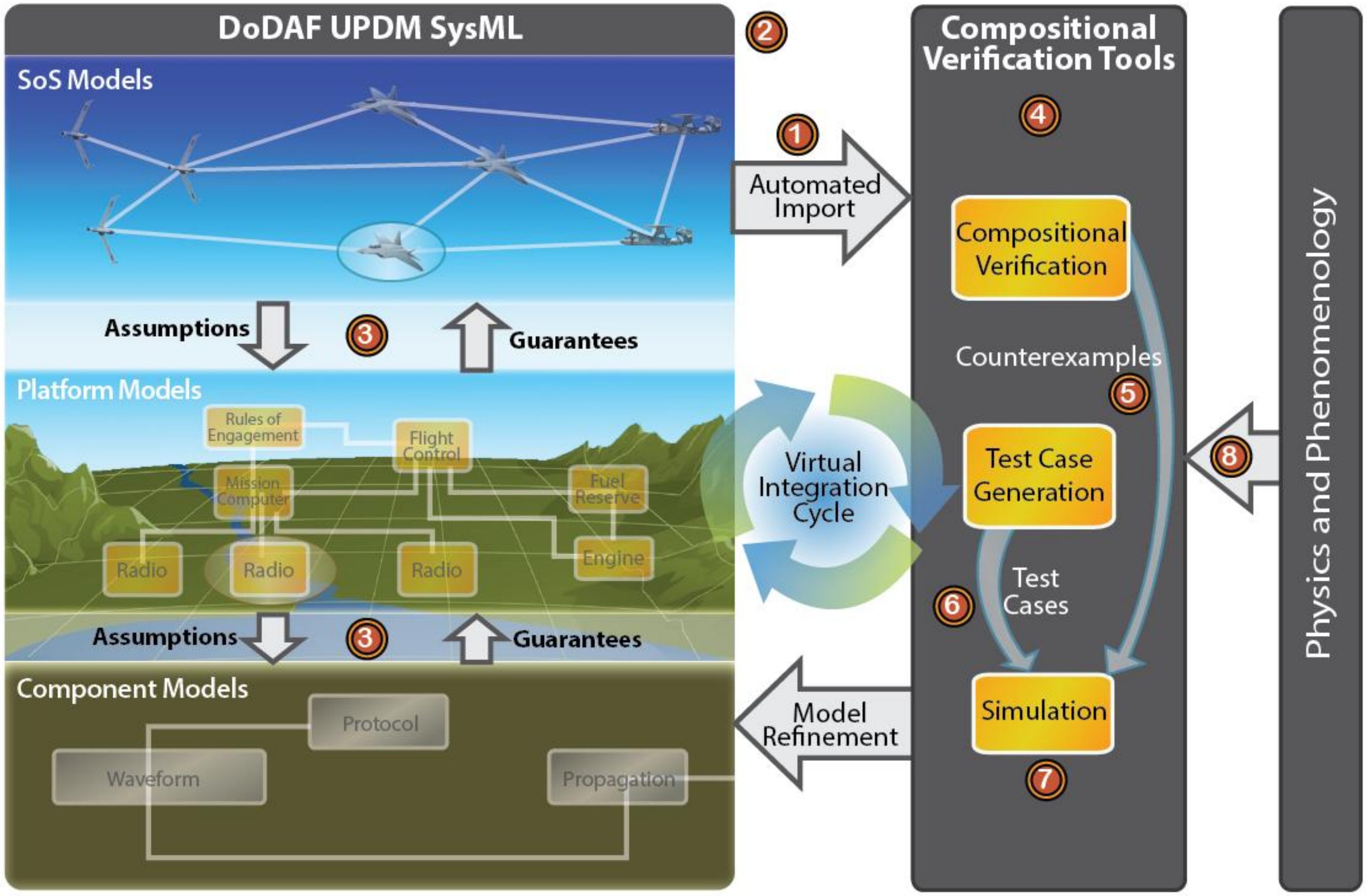
This is not entirely new

- Web services illustrate many of these properties
  - Excellent separation of mechanism from function
  - For well-architected systems, excellent horizontal scalability
  - Straightforward interoperability through shared protocols
  - Strong CI practices in industry



- Substantial improvement on the **deployment architecture** of previous systems
- However: few constraints on Size, Weight, and Power; Latency is unpredictable; security is uncertain, etc.

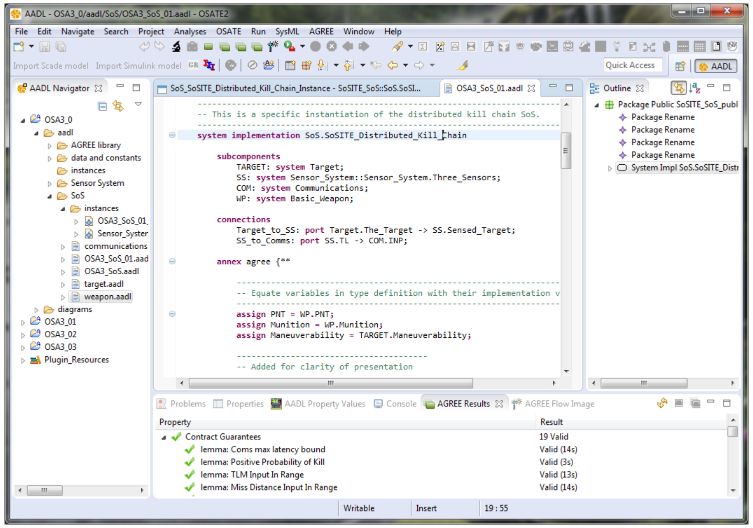
Towards Integration Supported By Proof



1. *Automated import of DoDAF compliant models*
2. *Architecture driven approach scales formal verification to SoS*
3. *Assume/guarantee contracts enable analysis of heterogeneous and legacy components*
4. *Single formal model used for verification, test generation, and simulation*
5. *Formal verification finds errors before deployment*
6. *Metric-based test generation provides known coverage of the SoS and legacy components*
7. *Simulation enables debugging of counterexamples and early validation of contracts*
8. *Support for analysis of SoS physics and phenomenology*

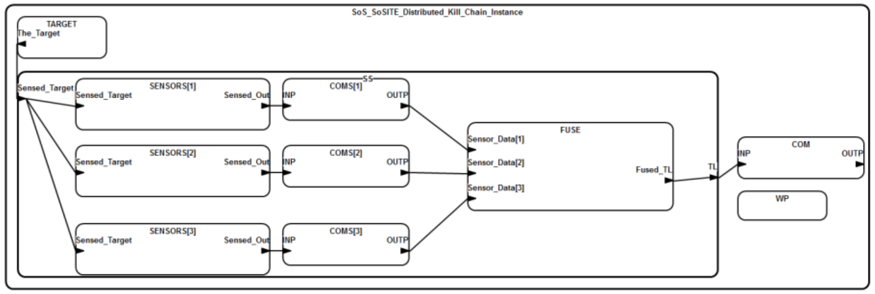
Architectural Modeling Using AADL

- Architecture Analysis & Description Language (AADL)
  - SAE Standard for Modeling System Architectures
- Developed under DARPA DSSA Program for U.S. Army Applications
- Textual and Graphical Modeling Notation



```
    ---------------------------------------------------------------
    -- This is a specific instantiation of the distributed kill chain SoS.
    ---------------------------------------------------------------
    system implementation SoS.SoSITE_Distributed_Kill_Chain

        subcomponents
            TARGET: system Target;
            SS: system Sensor_System::Sensor_System.Three_Sensors;
            COM: system Communications;
            WP: system Basic_Weapon;

        connections
            Target_to_SS: port Target.The_Target -> SS.Sensed_Target;
            SS_to_Comms: port SS.TL -> COM.INP;

        annex agree {**

            ---------------------------------------------------------
            -- Equate variables in type definition with their implementation v
            ---------------------------------------------------------
            assign PNT = WP.PNT;
            assign Munition = WP.Munition;
            assign Maneuverability = TARGET.Maneuverability;

            -------------------------------------
            -- Added for clarity of presentation
```

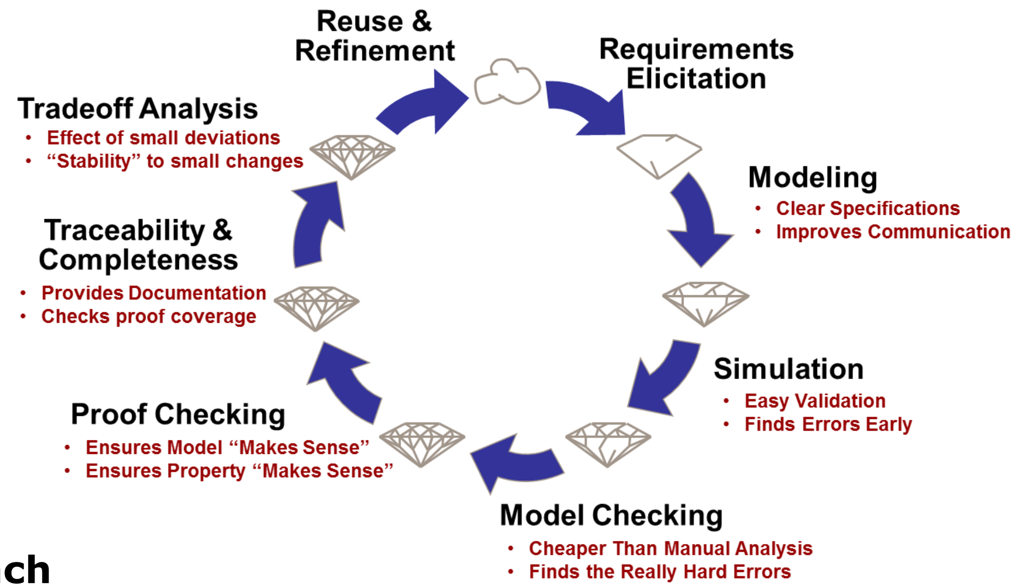
Property	Result
Contract Guarantees	19 Valid
lemma: Coms max latency bound	Valid (14s)
lemma: Positive Probability of Kill	Valid (3s)
lemma: TLM Input In Range	Valid (13s)
lemma: Miss Distance Input In Range	Valid (14s)

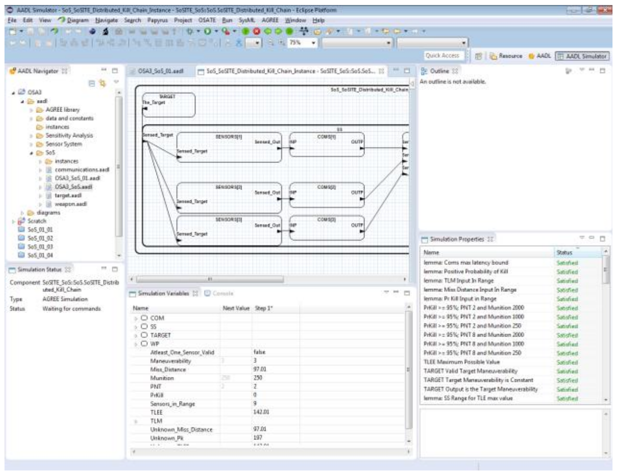


- Open Source Tools (OSATE) Provided by Software Engineering Institute (SEI)
- Semantics More Completely Defined than SysML
- Contains Embedded System Constructs that SysML Lacks
  - Supports non-functional analyses of architectures (performance, fault propagation)
- Extensible Through Annexes: our contract language is called AGREE

Towards a "Capability IDE" and System Engineering Process

- You are not done when you get a proof
- A good specification environment IDE needs:
  - Symbolic Simulator (exploration)
  - Compositional proof
  - Contract checker (realizability)
  - Specification checker (vacuity)
- Analogous to a good programming IDE
  - Editor, Debugger, Profiler, etc.
- SoSITE progress: **Specifier's Workbench**
  - Symbolic simulator
  - Automated traceability support
  - Completeness analysis
  - In progress: System optimization (using Pete Manolios' Inez tool)
- Improvements driven by feedback from several projects
  - SoSITE: Systems-of-systems verification, parametric proof
  - HACMS, CVFCS, SwPI, CRP: System verification
  - MFD: subsystem testing and verification





Simulation Variables

Name	Next Value	Step 1*
COM		
SS		
TARGET		
WP		
Atleast_One_Sensor_Valid		false
Maneuverability	3	3
Miss_Distance		97.01
Munition	250	250
PNT	2	2
PrKill		0
Sensors_in_Range		9
TLEE		142.01
TLM		
Unknown_Miss_Distance		97.01
Unknown_Pk		197

Simulation Properties

Name	Status
lemma: Coms max latency bound	Satisfied
lemma: Positive Probability of Kill	Satisfied
lemma: TLM Input In Range	Satisfied
lemma: Miss Distance Input In Range	Satisfied
lemma: Pr Kill Input in Range	Satisfied
PrKill >= 95%; PNT 2 and Munition 2000	Satisfied
PrKill >= 95%; PNT 2 and Munition 1000	Satisfied
PrKill >= 95%; PNT 2 and Munition 250	Satisfied
PrKill >= 95%; PNT 8 and Munition 2000	Satisfied
PrKill >= 95%; PNT 8 and Munition 1000	Satisfied
PrKill >= 95%; PNT 8 and Munition 250	Satisfied
TLEE Maximum Possible Value	Satisfied
TARGET Valid Target Maneuverability	Satisfied
TARGET Target Maneuverability is Constant	Satisfied
TARGET Output is the Target Maneuverability	Satisfied
lemma: SS Range for TLE max value	Satisfied

Composability and Complexity

- Current architecture description languages (SysML, AADL) and tools intermix **functional** and **deployment architecture**
  - Explicit "bindings" of functionality to threads, processes, physical resources
  - Makes architectures rigid, makes proofs complex
  - [Fred Brooks] separating essential complexity from accidental complexity

- Deployment should be automated
  - Non-functional performance parameters drive deployment binding process
    - Iterative binding generation throughout development cycle
    - No surprises
  - ADL support for synthesis and pre-verified architectural patterns
  - Richer ADLs look more like programming languages

- Multiple, independent analyses for functional and non-functional aspects
  - Capability proofs depend on schedulability, isolation, etc.
  - Principled mechanism for "passing the buck"

Challenges

- Composition requires **isolation**
  - Interactions must be specified in terms of interfaces
  - Software can be rife with hidden interactions:
    - Memory boundary violations
    - Starvation/misuse of resources
- Composition requires **trust boundaries**
  - Which components can I trust and how much?
  - Encryption, Authentication and non-repudiation of communications
  - C.f.: Charlie Miller Jeep car hack (Chrysler: 1.4M vehicle recall)
  - Middleware can become "switchboard" exposing all data
- Analysis requires **model fidelity**
  - HACMS: "fly what you analyze"
- Promising new technology
  - seL4 (Data61) microkernel with proofs of memory non-interference
  - Ivory / Rust programming languages ensure memory safety
  - Recent work in homomorphic encryption allows data translation without breaking encryption!

Recap

- Current organizational focus is on **parts** (components)
- Integration focus is on **wiring** and **static architectures**
- Focus should shift to **capabilities** and **behavior**
  - **Fail Early**
  - Determine KPPs and model them into architecture
  - Analytic rigor should be used
    - proofs when possible
  - Testing is too late, too expensive, and too incomplete
- Deployment architecture should be iterative and semi-automated
  - Increasing use of synthesis
  - Deploy throughout design cycle
- Systemic approaches should be used to ensure isolation
  - Partitioning, microkernels, and memory-safe languages