Architecture-led Incremental System Assurance (ALISA) Demonstration

Peter Feiler

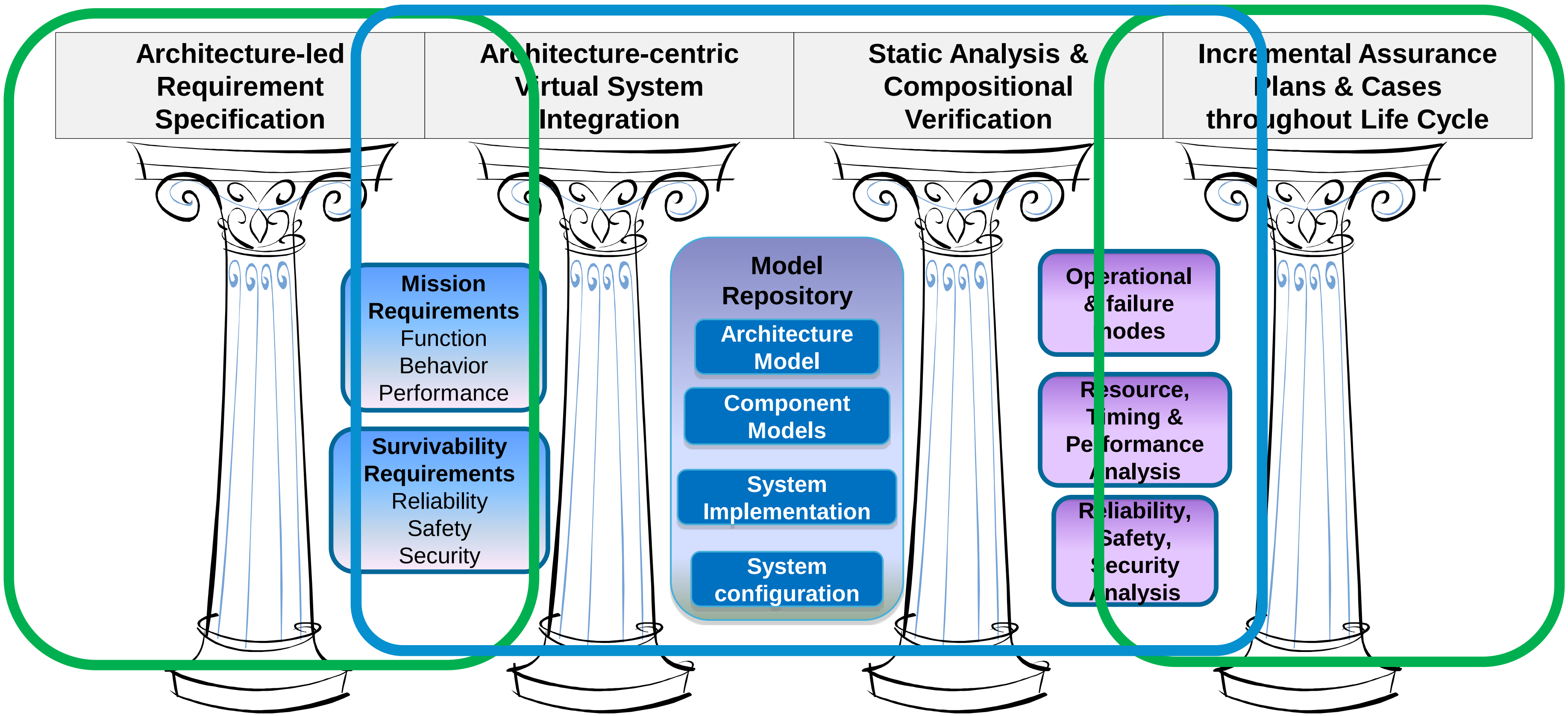
Feb 2017

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Assurance & Qualification Improvement Strategy

Assurance: <u>Sufficient</u> <u>evidence</u> that a <u>system implementation</u> meets <u>system requirements</u>

2010 SEI Study for AMRDEC Aviation Engineering Directorate



Architecture-centric Virtual System Integration (ACVIP)
Incremental Lifecycle Assurance (ALISA)

Value of Requirement Uncertainty Awareness

Textual requirement quality statistics

- Current requirement engineering practice relies on stakeholders traceability and document reviews resulting in high rate of requirement change.

Requirements error	%
Incomplete	21%
Missing	33%
Incorrect	24%
Ambiguous	6%
Inconsistent	5%

NIST Study

Managed awareness of requirement uncertainty reduces requirement changes by 50%

- 80% of requirement changes from development team
- Expert requirement uncertainty assessment
  - Volatility, Impact, Precedence, Time criticality
- Focus on high uncertainty areas
- Engineer for inherent variability

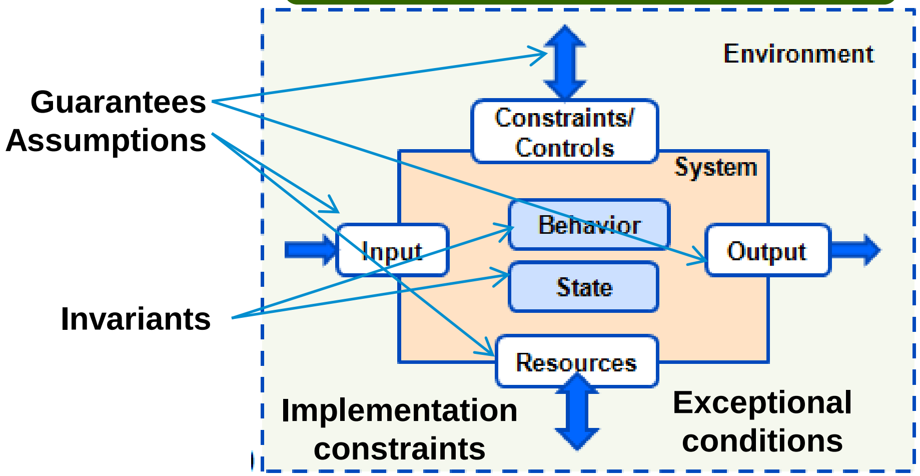
Selection	Weight	Precedence
Low Precedence	9	No experience of concept, or environment. Historically volatile
Medium Precedence	3	Some experience in related environments. Some historic volatility
High Precedence	1	Concept already in service. Low historic volatility

Figure 8. Precedence measurement scale

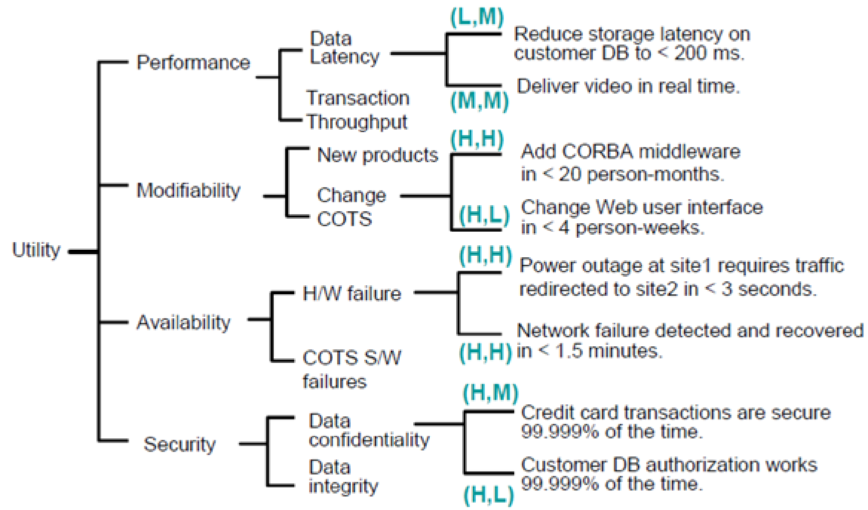
Rolls Royce Study

Three Dimensions of Requirement Coverage

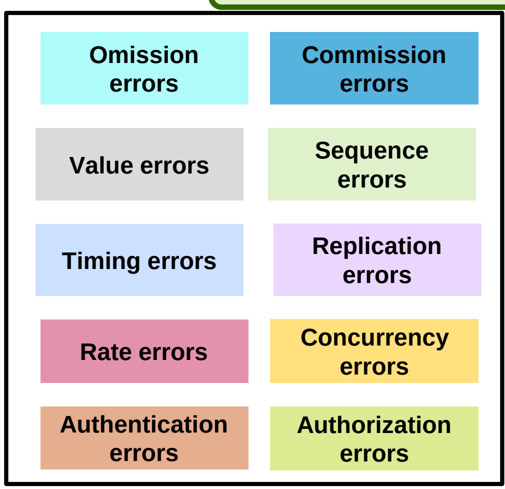
System interactions, state, behavior



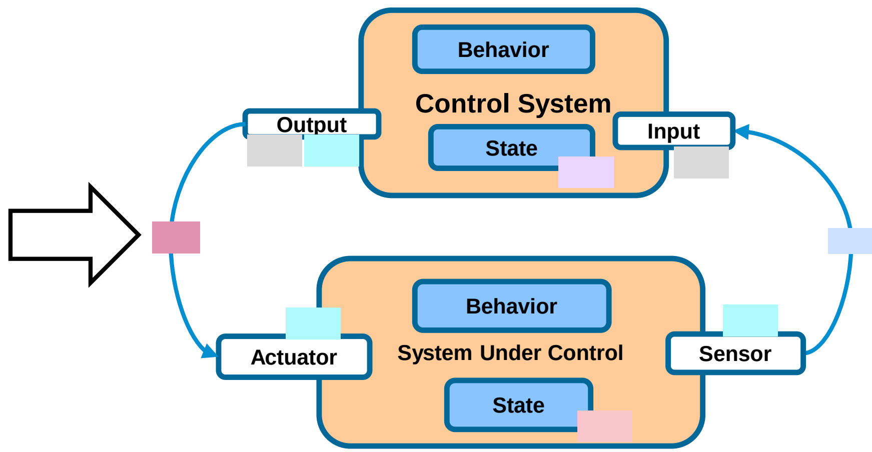
Design & operational quality attributes



Fault impact & contributors

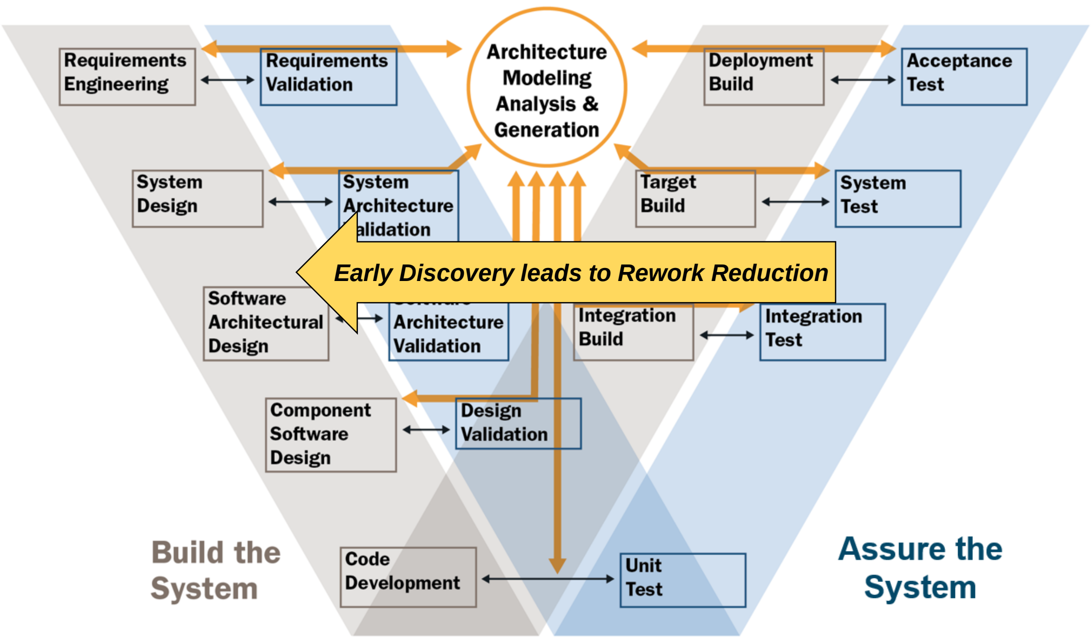


Fault Propagation Ontology



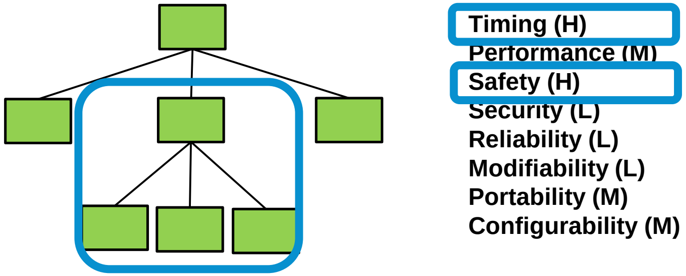
Three Dimensions of Incremental Assurance

Incremental assurance through virtual system integration for early discovery
Return on Investment study by SAVI*

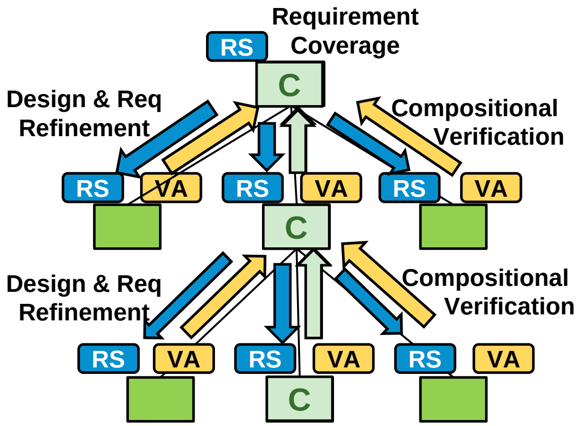


*System Architecture Virtual Integration (SAVI) Aerospace industry initiative

Priority focused architecture design exploration for high payoff
Measurable improvement (Rolls Royce)



Compositional verification and partitions to limit assurance impact



Modeling Notations in ALISA Prototype

ReqSpec: Represent stakeholder goals, system requirements
- Document-based & architecture-led
- Verifiable system requirements
- Coverage and uncertainty

Verify: Verification plans of verification activities against artifacts
- Reasoning logic of how verification activities satisfy requirement
- Via verification methods (manual, automated) on models/code
- Assumptions, preconditions on verification methods

Alisa: Composition of verification plans into assurance cases
- Verification of AADL model artifacts
- Across layers of system architecture
- Assurance tasks as filtered views of assurance plans

Assure: Manage assurance case instance execution and results
- Multi-valued logic evaluation of verification action & results
- Acceptable risk factors (e.g., design assurance levels)
- Filtered execution of assurance plans (based on category tags)

ReqSpec: Textual Notation for Draft RDAL Annex

Objective

- Goal-oriented architecture-led requirement specification
- Explicit record of requirement decomposition, refinement, and evolution
- Integration with safety analysis via Error Model Annex V2
- Basis for incremental life-cycle assurance of evolving systems
- Bridge to existing requirements documentation practices

Stakeholder Goals

- Descriptions of intent that may be in conflict with each other
- Use cases: UML/SysML or ITU UCM

System Requirement Specification

- Specification of a consistent set of system requirements that must be met by a delivered system
- Basis for verification and assurance plans

Use Scenarios

External stakeholder requirement documents

- Leave external: requirements reference document entry (**see document**)
- Import as goal document
- Capture concepts, functions, and architecture elements in AADL model
  - Basis for a system requirement specification

Use Scenarios

External system requirement documents

- Leave external
  - Capture concepts, functions, and architecture elements in AADL model
  - Create reqspec requirement proxies with reference to document entry (**see document**)
  - Annotate requirement with model reference (**for**)
- Import as requirement document
  - Capture concepts, functions, and architecture elements in AADL model
  - Annotate requirement with model reference (**for**)

Resulting ReqSpec specification is basis for verification and assurance plans

Stakeholder Goals Notation

Goal specification

- Associated with AADL model element
- Description, rationale
- Refinement, evolution, conflict resolution
- Traceability to stakeholders and DOORS requirement and other documents

- Document focused goal organization
  - Goals in document sections
  - Incrementally evolve model and identify model elements
- Model focused goal/requirement organization
  - Requirements for a model component

Names, Concepts, and Systems

```
document SCSgoals : "SCS stakeholder goals" [
    description "This document contains the stakeholder requirements for the Simple Control System (SCS)
The SCS provides control for a simple device (SD)
The SCS system consists for software, hardware, and physical components."
    section SystemFunctionality : "System Functionality" [
        goal g1: "Feedback Control"
        [ description "The simple controller (SC) shall provide stable feedback control of the SD."
        rationale "The SD is a safety critical device that cannot tolerate erratic behavior."
        stakeholder sei.phf
        ]
        goal g2: "Digital Feedback Control" [
            description "The SCS system shall control the SCS device with a digital controller"
        ]
        goal g3: "electrical power"
        [ description "The simple control system shall be supplied with 15V electrical power."
        stakeholder sei.phf
        ]
    section NonfunctionalProperties : "Nonfunctional system requirements"
    [
        goal ng1 : "Safety" [
            description "The system shall be safe"
            rationale "This is a control system, whose failure affects lives. "
            stakeholder sei.phf sei.dpg
        ]
        goal ng1_1: "Physical damage"[
            refines ng1
            description "The controller shall not cause the simple device to damage objects
        ]
```

Are the following the same:

Simple Control System

Simple Controller

System

Controller

We define SCS as system type.

As we add requirements we add features, e.g., power.

As we define a model of the operational context we are explicit about the necessary and possibly missing interactions.

Mixture of Requirements & Architecture Design Constraints

Textual Requirements for a Patient Therapy System

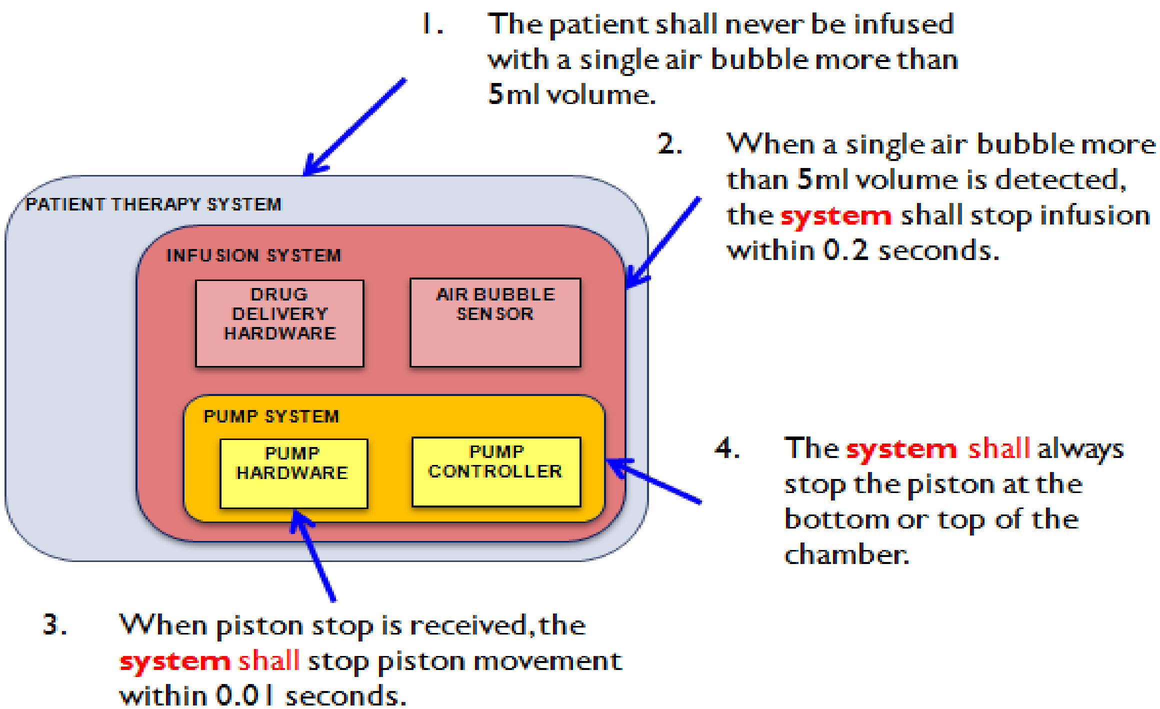
The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the **system** shall stop infusion within 0.2 seconds.

When piston stop is received, the **system** shall stop piston movement within 0.01 seconds.

The **system** shall always stop the piston at the bottom or top of the chamber.

Same Requirements Mapped to an Architecture Model



Importance of understanding system boundary

We have effectively specified a system partial architecture

U Minnesota Study

System Requirement Specification Notation

Requirement specification

- Verifiable contract via verification plans
  - Satisfiable design goal
- Associated with AADL classifier or element in classifier
- Description, rationale
- Parameterization by variable
- Refinement, decomposition, inheritance, evolution
- Verifiable by verification activities
- Formal specification of predicate
- Traceability to goals, hazards
- Traceability to DOORS requirement and other documents

Requirement Traceability Reports

BIRT based prototype

- Traceability to stakeholders
- Traceability between requirements and goals
- Context command invoked on instance model, reqspec/goal file
- Source code available as org.osate.reqtrace
- Extensible for additional report templates

ReqSpec compiler validations

- Reported via Eclipse Diagnostics/Markers
  - Missing stakeholder, missing goal, duplicate goal/requirement
  - Cycles in refine hierarchy
- Report generation from Eclipse Markers

Requirement Sets

System Requirement Set

- One per component classifier (identified by **for** clause)
- List of system requirement declarations
- Inherited requirements according to extends hierarchy
- Inclusion of individual global requirements or global requirement sets

Global Requirement Set

- Reusable set of requirements
- For inclusion in system requirement sets
- Configurable into assurance plans

User-extensible Categories

Requirement categories for filtering and coverage

Can be organized into category sets

predefined.cat

```
Quality [
    Behavior State Performance
    Latency Timing Security Safety Reliability Availability
    CPUUtilization MemoryUtilization
    NetworkUtilization  Mass ElectricalPower
]
Phase [
    StakeholderRequirements SystemRequirements PDR CDR
    ArchitectureDesign DetailedDesign Implementation UnitTest
    SystemTest
]
Layer [
    Tier1 Tier2
    Tier3 Tier4 Tier5
]
Kind [
    Assumption Guarantee Consistency Constraint Exception
]
```

Minimize Impact of Frequent Requirement Changes

```
system requirements scsreqs for SimpleControlSystem::SCS [
    val MaximumLatency = 20 ms
    requirement R1 : "SCS weight limit" [
        val  MaximumWeight = 1.2 kg
        category Quality.Mass
        description this " shall be within weight of " MaximumWeight
        // second condition verifies that MaximumWeight is same as the property value WeightLimit
        value predicate MaximumWeight == #SEI::WeightLimit
        see goal SCSgoals.ng2
    ]

    requirement R2 : "SCS sensor to actuator response time limit" [
        description this " shall have a sensor to actuator response time within " MaximumLatency
        category Quality.Latency
        see goal SCSgoals.g1
    ]
```

Val constants can be defined for requirement set or individual requirements.

Val constants can be used in descriptions, predicates, and as parameters in verification activities

```
verification plan scsvplan for scsreqs
 [
     claim R1 [
         activities
             actualsystemweight : Plugins.MassAnalysis()
                 [ category Quality.Mass ]
             MaxWeight : Resolute.verifySCSReq1(MaximumWeight in kg) [ category Quality.Mass ]
     ]
```

AADL Model as System Specification

```
system SCS
    features
        power: in feature PhysicalResources::Power;
        force: out feature;
    modes
        operational: initial mode;
        standby: mode;
    properties
        ACVIP::Aliases => ("SCS", "Simple Controller");
        SEI::PowerBudget => 2.5 w applies to power;
        SEI::WeightLimit => 1.2 kg;
        Physical::Voltage => 12.0V applies to power;
    annex emv2 {**
            use types ErrorLibrary;
            use behavior ErrorLibrary::FailStop;
            error propagations
                force: out propagation {ServiceOmission};
                flows
                    es1: error source force;
            end propagations;**};
end SCS;
```

```
value predicate MaximumWeight == #SEI::WeightLimit
```

Scenario 1

Property association value changed by user

Referenced in description instead of val **variable**

Scenario 2

Val **variable value automatically set as property value for certain verification methods**

Scenario 3

Consistency between val **variables and property values**

Consistency check: explicit or as part of verification method registry

Shared Predicates and “Compute” Variables

```
requirement R3 : "SCS inlet voltage" for power [
    val volts = 12.0
    compute actualvolt: real
    value predicate volts == actualvolt
    see goal SCSgoals.g3
]
```

Think of “compute” variable as free variable in lambda expression.

```
claim R3  [
    activities
        hasvoltage: Alisa_Consistency.ConsistentVoltage(volts)
        consistentvoltage: actualvolt = Alisa_Consistency.GetVoltage()
    ]
```

verification activity result is bound as compute value

```
method ConsistentVoltage (feature, voltage: real ) boolean
:"Ensure Voltage property value is consistent with required voltage value" [
    java alisa_consistency.ModelVerifications.hasVoltage(String name, double voltage)
    description "Verify that the Voltage property has the same value as specified in the requirement"
]

method GetVoltage (feature) returns (volts: real )
:"Ensure Voltage property value is consistent with required voltage value" [
    java alisa_consistency.ModelVerifications.getVoltage
    description "Verify that the Voltage property has the same value as specified in the requirement"
]
```

Global Requirements

```
global requirements globalReq
[
    requirement connected : "All features of a component are connected"[
        description "All features of a component are connected."
        when alisa_consistency.ModelConditions.isLeafComponent()
        category Kind.Consistency
        development stakeholder sei.phf
    ]

    requirement Allconnected : "All features of all components are connected"[
        description "All features of leaf components are connected."
        category Kind.Consistency
        development stakeholder sei.phf
    ]
]
```

Reusable requirements, typically consistency constraints and completeness checks.

```
include Peter.req1 for self
include globalReq.Allconnected for self
include globalReq.connected
```

Inclusion in system requirement sets.

Applicable for component itself or recursively for all subcomponents.

Allconnected: the verification method is assumed to recurse the component hierarchy.

Connected: the requirement is applied only to components that satisfy when **condition**

Requirements Decomposition

Derive requirements on subcomponents from system requirements

- Associate requirement with subcomponent in the system implementation
  - Context specific requirement
  - Requirements on subsystem to be acquired (purchased, contracted out)
- Requirement on classifier of subcomponent
  - applies to all subcomponent instances referencing the classifier
  - Represented by property of subcomponent classifier
  - System specification used in implementation to verify that design meets system requirement

Verification & Assurance Plan Notation

Method registry

- Reusable verification methods on models and other artifacts
- Method categories for filtering and coverage
- Currently supported: Java, Resolute, analysis plugin, manual

Verification plans

- Claims aligned with system requirement specifications
- Multiple verification activities on different artifacts
- Implementation and use verification
- Conditional and backup activities (pass, fail, error)
- User definable selection categories
- Description, Rationale

Compositional assurance plans and tasks

- Configuration of assurance responsibility
- Focused assurance view based in category filters

Registry of Verification Methods

builtinmethods.verify

```
    method sumBudgets : "sum all weight budgets" [
        compute self path "org.osate.assure.analysis.PlatformResourceBudgets.sumSubB
            method hasWeightBudgets
        ] validation weightCoverage [
            method hasWeightBudgets
        ]
    ]
    method hasWeightBudgets : "Determine weight budget coverage" [
        compute self path "SubcomponentWeightBudgetCoveragePercent"
    ]
    method assertSubBudgets : "compare sum of all weights to limit of 0.5 kg" [
        predicate self path "org.osate.assure.analysis.PlatformResourceBudgets.asser
    ]
    method verifyR1 : "Assure R1" [
        resoluteprove all path "SCSReq1"
    ]
    method AllResponseTimes : "Verify all flow latencies in system instance" [
        analysis a
        descript
For details see
```

Outline

- builtinmethods
  - builtinmethods
    - sumBudgets
      - www
      - weightCoverage
    - hasWeightBudgets
    - assertSubBudgets
    - verifyR1
    - AllResponseTimes
    - ComponentOwnedFlowResponseTimes
    - SingleFlowResponseTime
    - ElectricalPower
    - ConnectionBinding
    - A429Mapping
    - PortDataContracts
    - ...is
    - ...WeightConsistency
    - ...rossNetWeightConsistency

Verification Method Registry Implementation

Existing OSATE Analysis plugins

Resolute claim functions

Reflective execution of Java/Xtend functions

Execution of Junit-based code tests

More to come (e.g., Agree, Simulink, SCADE, manual)

Why And/Or Assurance Logic is Insufficient

- Assurance case: focus on final result
  - Multiple pieces of evidence must be satisfied (“and”)
  - Design alternatives, different operational modes (“or”)
- Resolute experience during development
  - “and”: if first verification action fails, others are not evaluated
    - Equivalent to compiler stops on first error
  - “or”: any satisfied verification action is fine. Pick the cheapest one to execute
  - “implies”: failing first verification action results in success

Verification Activities

Multi-valued verification activity results

- Verification activity result states
  - Success, fail, error, tbd
- Compositional Argument Expressions
  - **All** [va+]: a collection of independent Vas
  - Va1 **then** Va2: Execution of Va2 dependent on success of Va1
  - Va1 **else** Va2: Execute Va2 only if Va1 produces negative result
  - Va1 **else [fail:** Va21 **timeout:** Va22 **error:** Va23**]**

Mode specific verification activities

Parameterized verification activities

- Data sets as input parameters

Assurance Plans and Cases Execution

Assurance plan

- Configuration of assurance case:
  - how much of architecture to be verified,
  - which component specific and reusable verification plans to be included

Assurance plan instantiation & execution

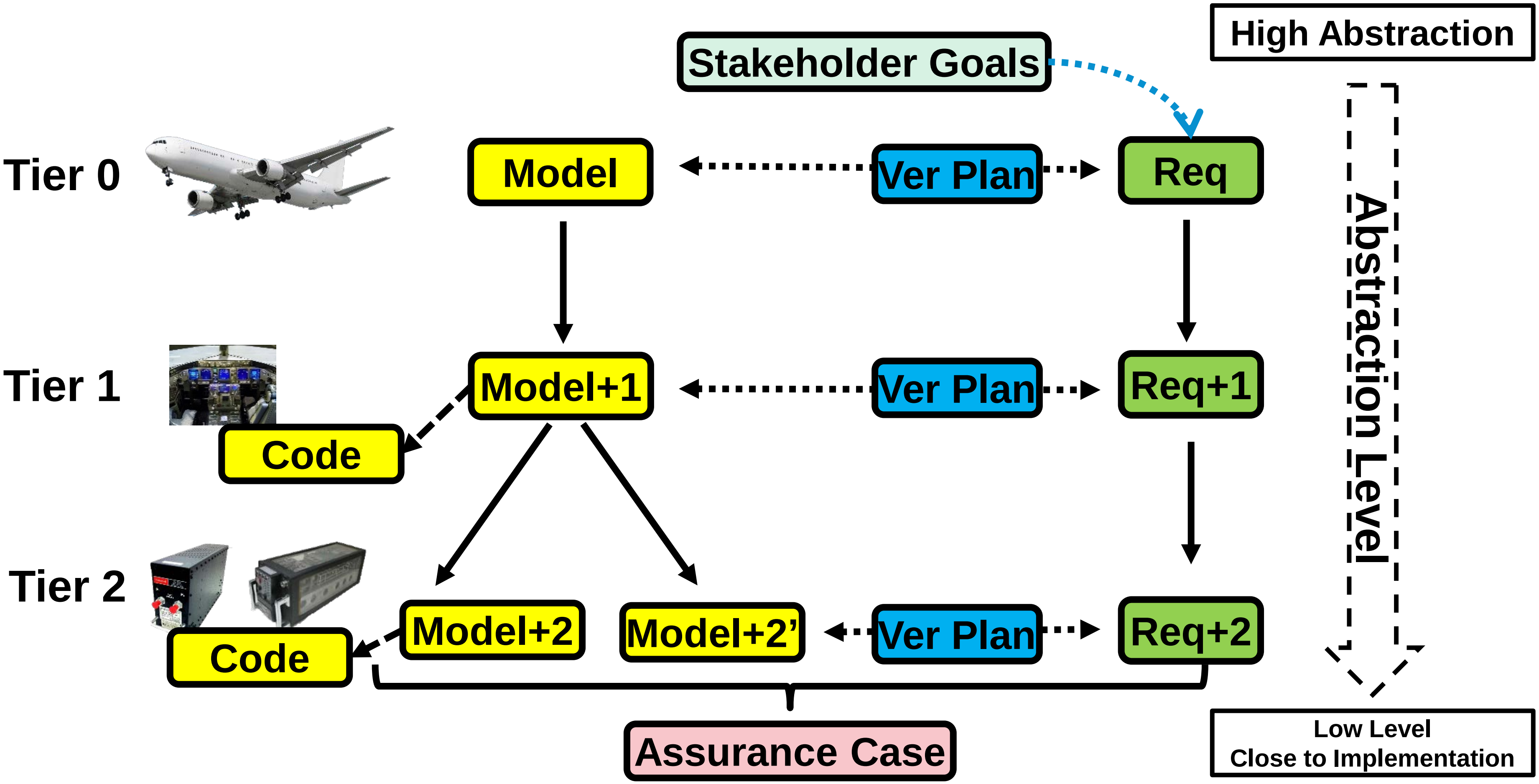
- Automated verification activity execution
- Execution states: To be done, in progress, done, Redo
- Tracking of result state and reports

Assurance Task

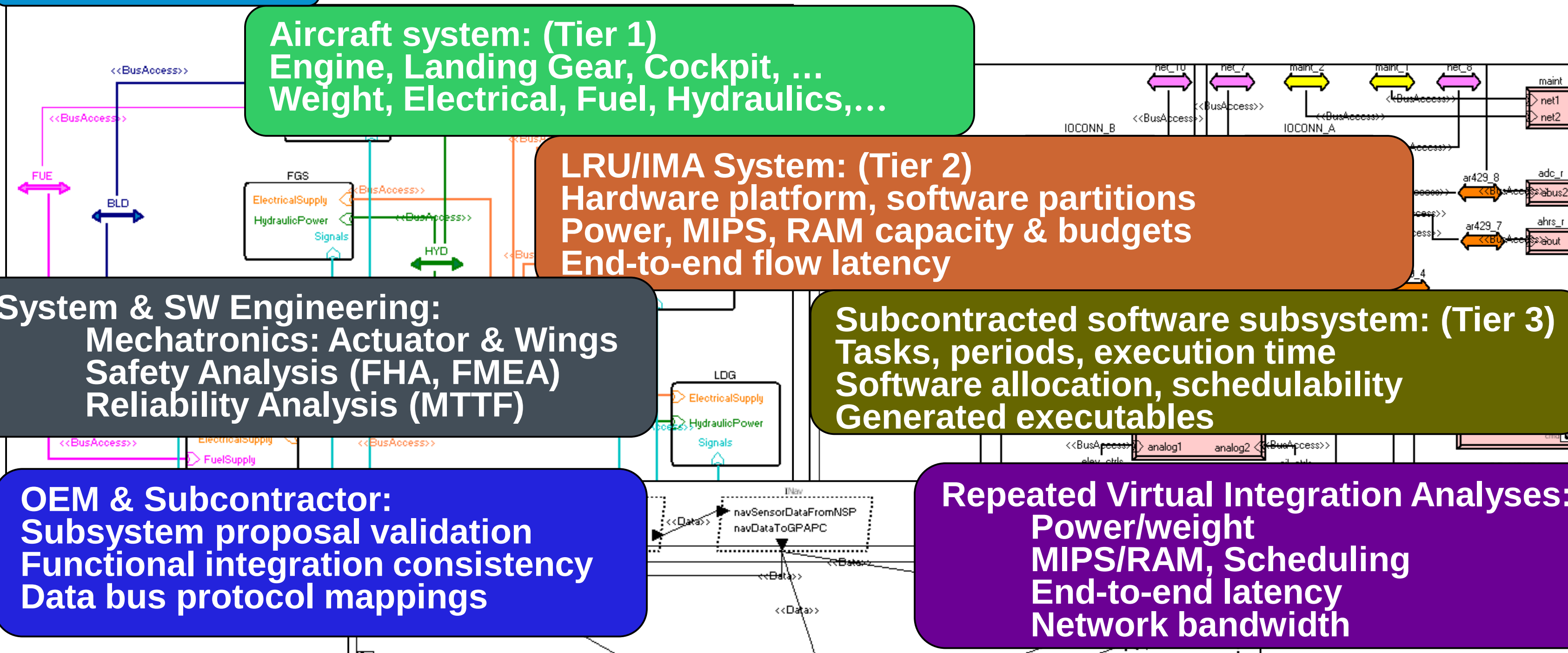
- Filtered assurance plan instances based on requirement, verification, and verification activity selection categories

Automated Incremental Assurance Workbench

Identify Assurance Hotspots throughout Lifecycle



Original Aircraft Virtual Integration Demo



Aircraft: (Tier 0)

Aircraft system: (Tier 1)
Engine, Landing Gear, Cockpit, …
Weight, Electrical, Fuel, Hydraulics,…

LRU/IMA System: (Tier 2)
Hardware platform, software partitions
Power, MIPS, RAM capacity & budgets
End-to-end flow latency

System & SW Engineering:
Mechatronics: Actuator & Wings
Safety Analysis (FHA, FMEA)
Reliability Analysis (MTTF)

Subcontracted software subsystem: (Tier 3)
Tasks, periods, execution time
Software allocation, schedulability
Generated executables

OEM & Subcontractor:
Subsystem proposal validation
Functional integration consistency
Data bus protocol mappings

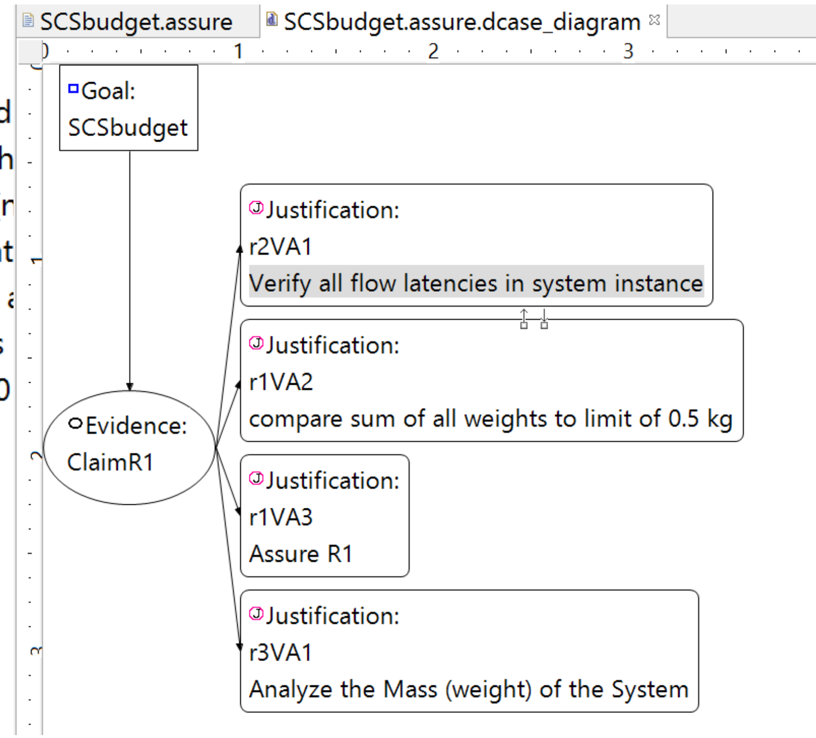
Repeated Virtual Integration Analyses:
Power/weight
MIPS/RAM, Scheduling
End-to-end latency
Network bandwidth

Proof of Concept Demonstration and Transition by Aerospace industry initiative
- **Propagate requirements and constraints**
- **Higher level model down to suppliers' lower level models**
- **Verification of lower level models satisfies higher level requirements and constraints**

- Multi-tier system & software architecture (in AADL)
- Incremental end-to-end validation of system properties

Assurance Results

System SaviDemo: (S9 F1 T0 E0 tbd0 EL0 TS0)
- ClaimR1: The weight of the Aircraft system shall not exceed 70000.0kg (S2 F0 T0 E0 tbd0 EL0
  - Evidence weightlimit: Perform full weight (mass) analysis. This includes net/gross weight con
    - top AircraftSystem.SubsystemSpec: [A] Sum of weights 67935.000 kg below weight limit 70
  - Evidence weightlimit2: Perform full weight (mass) analysis. This includes net/gross weight co
    - top AircraftSystem.SubsystemSpec: [A] Sum of weights 67935.000 kg below weight limit 70
- System FGS: (S6 F0 T0 E0 tbd0 EL0 TS0)
- System ELE: (S1 F1 T0 E0 tbd0 EL0 TS0)
  - ClaimR1: The weight of the Electrical system shall not exceed
    - Evidence weightlimit: Perform full weight (mass) analysis. Th
      - system ELE: [L] Sum of weights / Gross weight 75.000 kg (r
  - ClaimR2: The Electrical System shall be capable of handling at
    - Evidence powercapacity: Analyze Electrical power demands a
      - system ELE: ** ELE power budget total 24500.0 W exceeds
      - system ELE: budget total 24500.0 W within supply 25000.0



Assurance Result Metrics
Objective: Identify Assurance Hotspots
Aggregate success, fail, timeout, error, todo
Aggregate Else Success, then skip

Assurance Case Execution and Metrics

Assurance Metrics

- Requirement coverage measures
- Multi-valued verification result measures and their aggregates
  - Pass, fail, incomplete, conditions, backups
- Weighted requirement claims, verification activity results
  - Reflect importance, uncertainty, effectiveness

Guidance throughout life cycle

- Measurement based uncertainty areas (hotspots)
- Change impact prediction

References

ALISA

Online documentation: https://rawgit.com/osate/alisa/develop/org.osate.alisa.help/contents/00-Main.html

Paper: http://www.erts2016.org/uploads/program/paper_13.pdf

ReqSpec

SEI TR: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=464370

JMR Shadow project: http://resources.sei.cmu.edu/asset_files/specialreport/2015_003_001_447187.pdf

Model Examples

https://wiki.sei.cmu.edu/aadl/index.php/Models_examples