Carnegie Mellon University Software Engineering Institute

Contracts in System Development: From Multiconcern Analysis to Assurance

MARCH 17, 2023

Jérôme Hugues
Sam Procter

Contracts in Software Engineering **and beyond**

Well established practice to support (de)composition and analysis

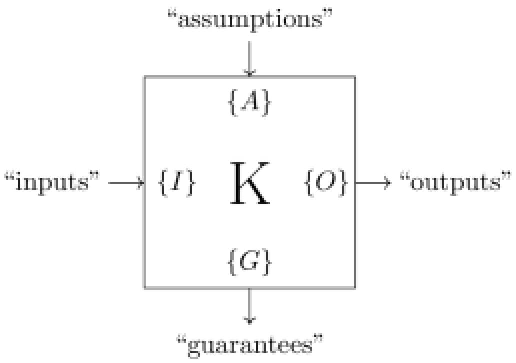
Pre/post conditions (Eifflel, then SPARK2014, ACSL, …)

```
set_second (s: INTEGER) -- Set the second from `s'.
require
        valid_argument_for_second: 0 <= s and s <= 59
ensure
        second_set: second = s
end
```

Assumptions/guarantees on component interfaces (10.1016/j.scico.2017.12.007)

Definition (Contract). A contract related to an element is a tuple K=(I,O,A,G):

- I are **inputs**: the data required by the element,
- O are **outputs**: the data provided by the element,
- A are **assumptions**: the properties required by the element,
- G are **guarantees**: the properties provided by the element.



Journal First: *Contracts in System Development: From Multi-Concern Analysis to Assurance, J. Hugues and S. Procter*

IEEE Software: 10.1109/MS.2022.3167533

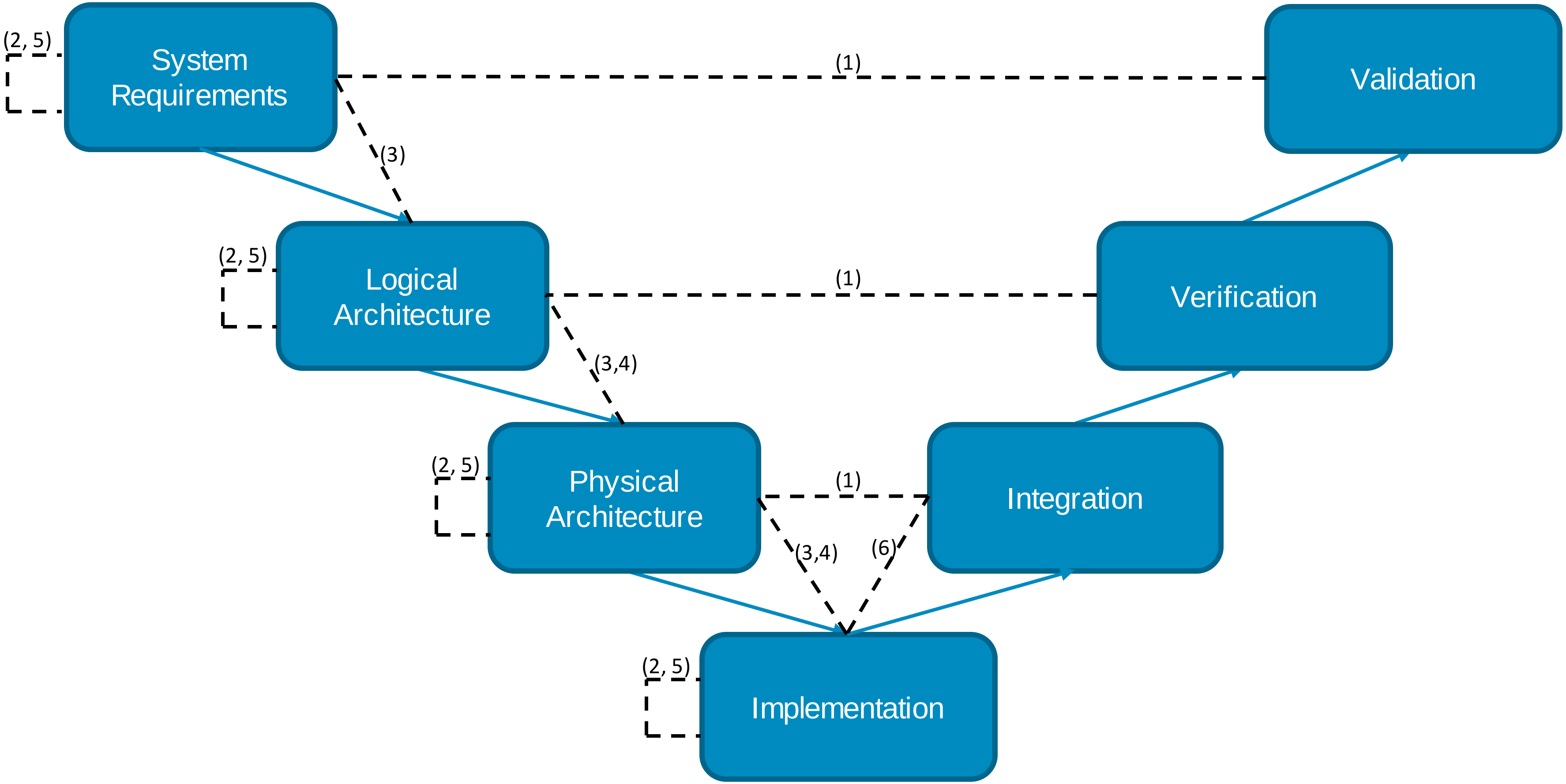
Short (5 pg) position paper, mostly

Contribution: metaphor of contract applies beyond system design.

- Common elements: two elements (parties), definition of a property to agreed upon and verification method that evaluates the property
- ⇒ Review of a typical V-cycle for applicability
- ⇒ “The House Believes that... formalized contracts improve processes definitions and reasoning”

State-of-the art: different contract types (e.g.: behavioral, interface, quality-of-service) support either the verification of the design (proper refinement, decomposition) or of different system properties (timing, confidentiality, etc.)

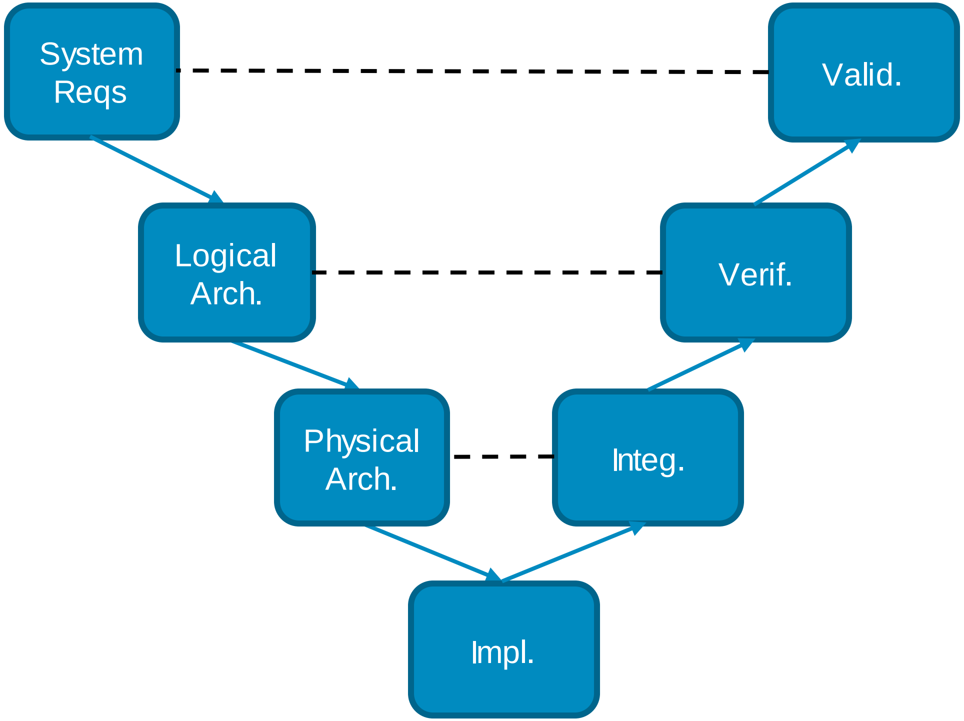
A notional system development process



[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

1: Requirement Allocation and Verification

a.k.a *“Am I delivering the right product?”*



Parties: “customer” , “Q&A”

Requirements are developed and allocated to system elements, recursively

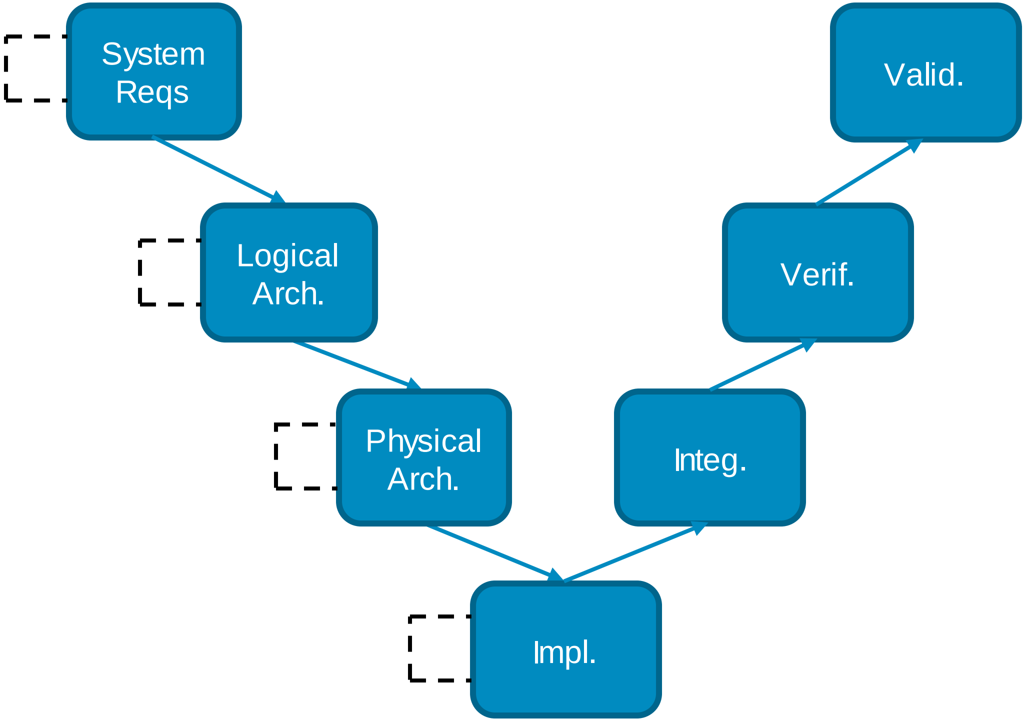
Requirements might be decomposed hierarchically and allocated

Examples:

- Contract meta-theory, Benveniste et al.
- Requirements associated with element(s)
- Attached to verification method (eg, review, JUnit test, formal methods, etc.)

2: Analysis Contracts

a.k.a *"Is verification Artifact A correct?"*



Parties: "model", "analysis tool"

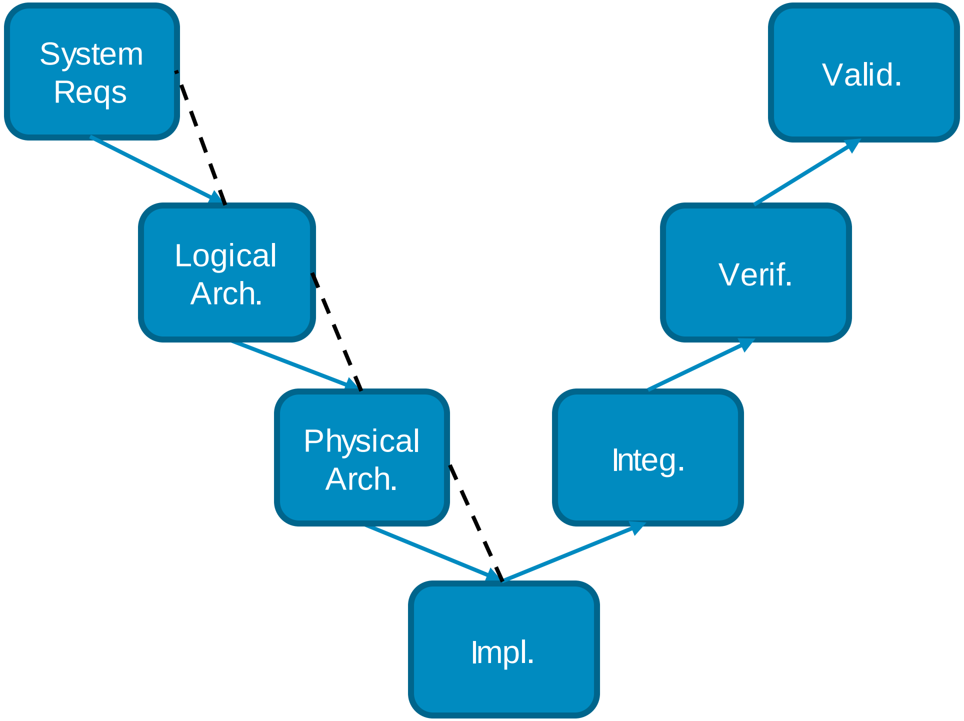
Often <u>implicit</u>, analysis contracts are assumptions made by an analysis on the model or implementation it operates on.

Evaluate whether an analysis can run and produce meaningful, correct output.

Why as a contract? Separation of concerns, tool vs applicability of a tool. Later part of assurance argument.

3: Vertical Integration Contracts

a.k.a "Is my decomposition correct?"



Parties: elements of system design

Assumptions and guarantees between models of the same system element at different levels of abstraction (refinement)

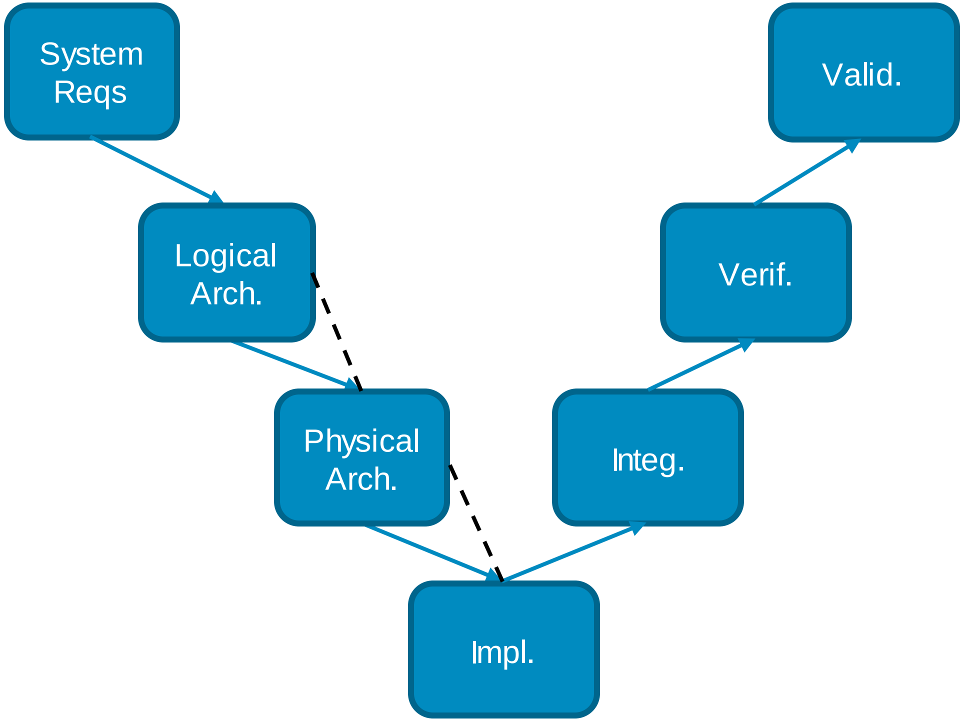
Often built into modeling language itself, notions of refinement and (de)composition

Contracts specified in terms of observable behaviors are verifiable using model checking or runtime verification

Example: observer pattern in Lustre, pre/post condition between spec and implementation

4: Horizontal Integration Contracts

a.k.a *“Is my (re)composition correct?”*



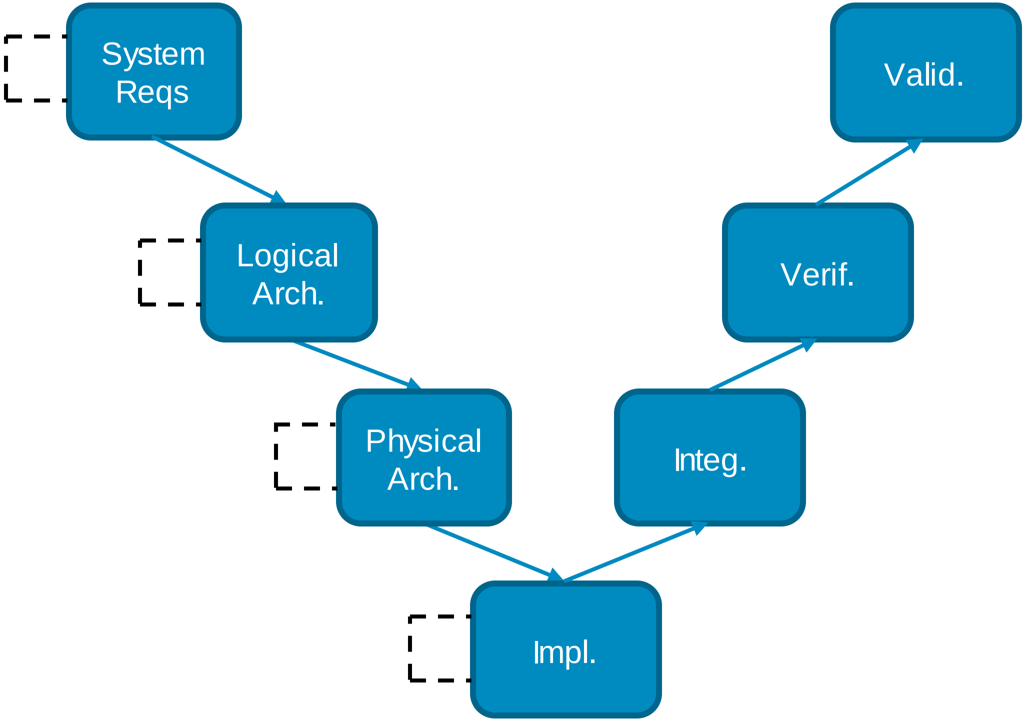
Parties: elements of system design

Dual of the precedent: composition rules for contracts allows to guarantee system-level properties from atomic elements.

Examples: pre/post conditions.

5: Conformance Contracts

a.k.a *“Am I building Artifact A correctly?”*



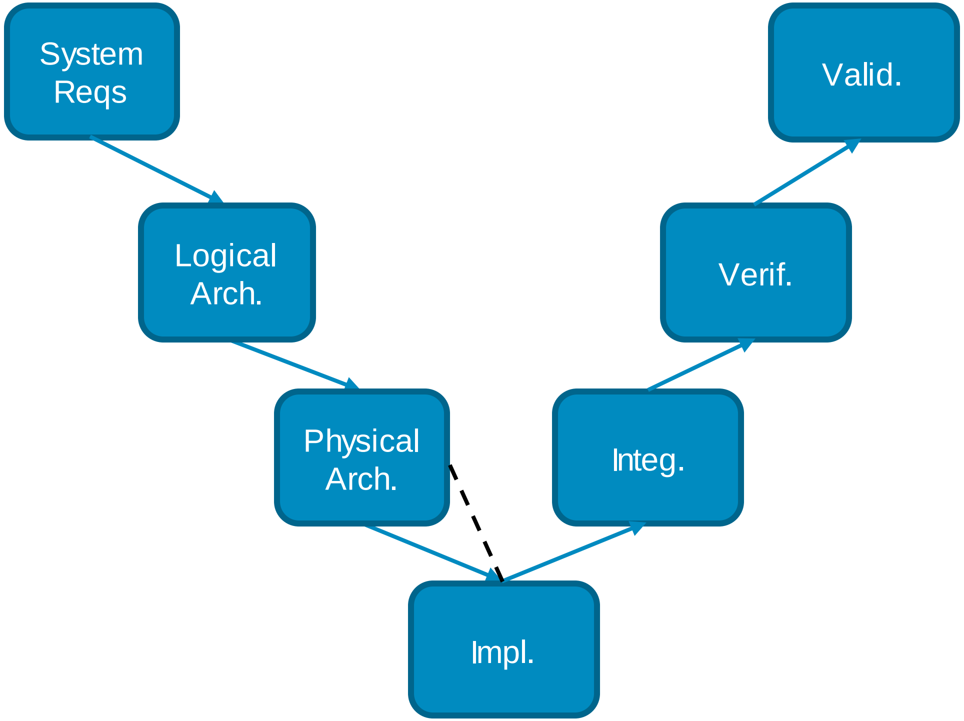
Many industries incentivize or require compliance with external standards or best practices

Similar to Analysis contracts, conformance contracts verify this compliance directly on system artifacts

E.g. conformance to architectural guidelines for coping with complexity, technical debt, or domain-specific constraints (e.g. safety-critical)

6: Implementation contract

a.k.a *“Does the implementation conform to its specs?”*



Special case of the vertical contract for the “big jump” between a model and the corresponding software source code

Two semantics gaps

1. Ideal semantics vs. software “physics”
2. Expression of a property in a model vs. as a test case

Conclusion – (What’s this about model-based? AADL?)

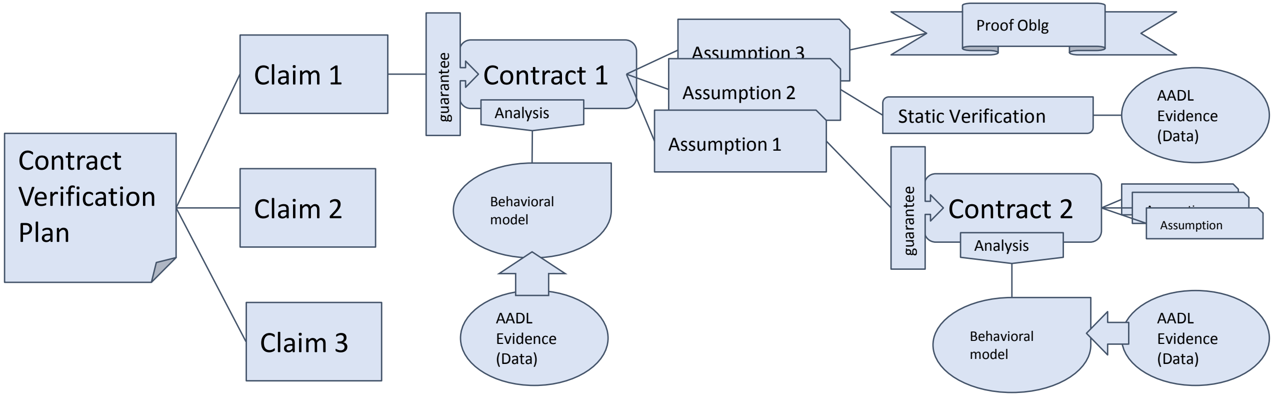
All these elements have been elaborated in the scope of the SAE AADL, a language for component-based design of safety-critical systems used for the last 20+ years

- ALISA (requirements, verification), Resolute/REAL (conformance, vertical), AGREE (horizontal)

Composing contracts enables <u>precise</u> system assurance: from top-level concerns to lower-level details

Full chain of custody

- Properties, assumptions,
- Verification methods used,
- Intermediate results,
- Rationale for composition, …


Contract Verification Plan
Claim 1
Claim 2
Claim 3
guarantee
Contract 1
Analysis
Behavioral model
AADL Evidence (Data)
Assumption 3
Assumption 2
Assumption 1
Proof Oblg
Static Verification
AADL Evidence (Data)
guarantee
Contract 2
Analysis
Assumption
Behavioral model
AADL Evidence (Data)