Assurance by Contracts in Cyber Physical Systems

Jüri Vain

Autumn 2017

Software Assurance

Lecture plan

- Last lecture was about JML and how it supports the specification of multi-view design contracts.
- Today we consider the contract-based specification of heterogeneous communicating components.
- The correctness conditions of contracts are discussed:
  - internal correctness of a contract
  - correctness of contracts compositions
- The approach is based on (formalization independent) ***contract meta-theory***

Quality requirements of Cyber Physical Systems

- **Informal Quality Requirements** are specified in the software requirements specification (*SRS*)
  - ***Real-Time Requirement*****:** The gate is closed when a train traverses the gate region, provided there is a minimal time distance of 40 seconds between two approaching trains.
    - ***Hard Real-time***: definite deadline specified after which system *fails.*
    - ***Soft Real-time***: after deadline specified quality of system's service *degrades.*
  - ***Safety Requirement*****:**

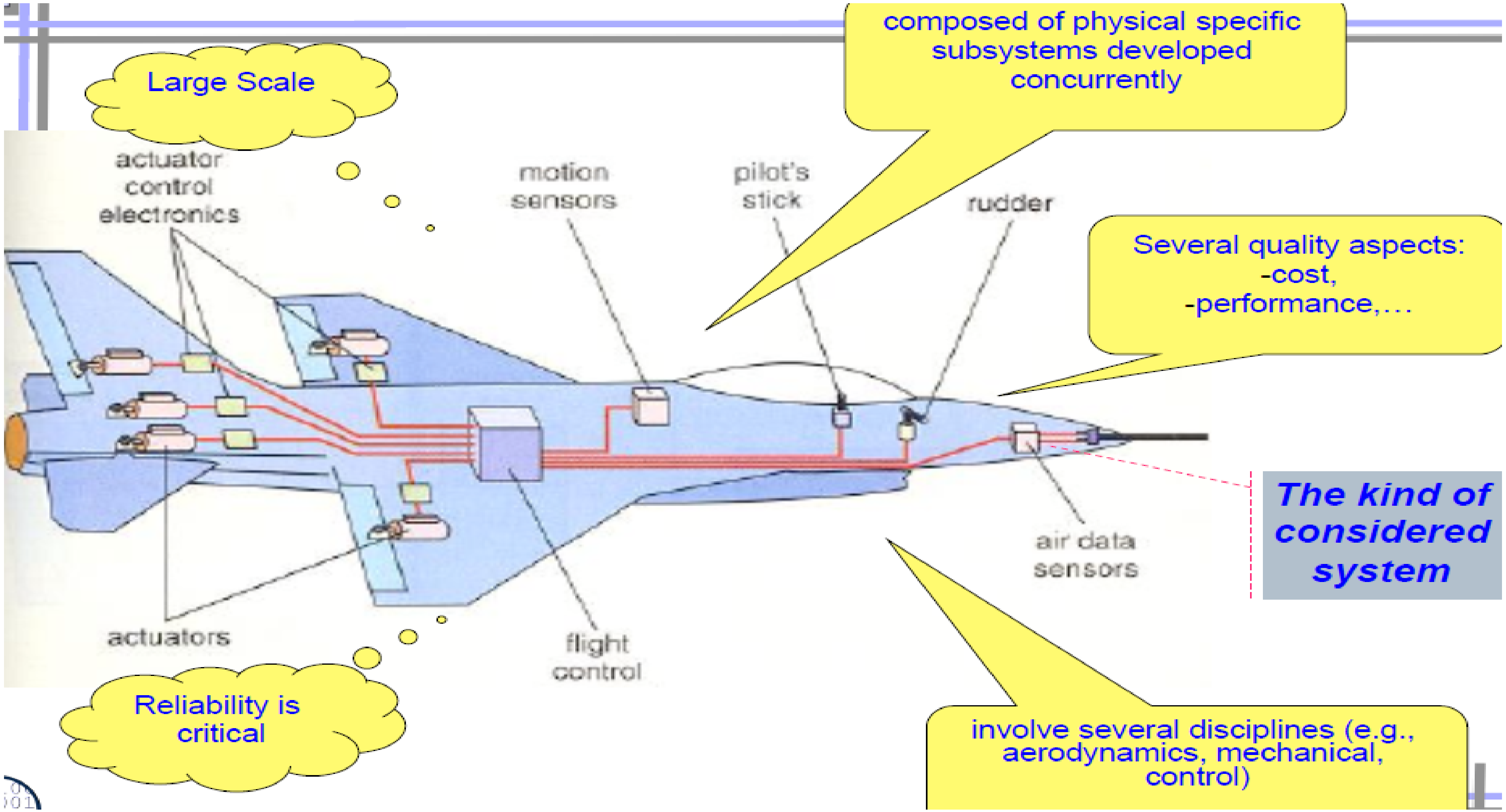
    If someone is between the train doors, the doors are kept open and train does not move
  - ***Energy Requirement***:

    If the robot's energy drops below 25% of the capacity of the battery, it should be able to reach the power plug to recharge.
  - ***Dynamic Movement Requirement***:

    If the car's energy sinks under 5% of the capacity of the battery, it will still be able to break and stop.

- All quality requirements can be expressed in terms of contracts.
- Underlying formal specification language must be **expressive enough** to capture the notions of the views the requirements concern.
- Contracts provide also the ***discipline*** of proving requirements correctness.
- The ***consistency*** (soundness) and ***relative*** completeness of requirements can be checked already before detailed design starts.

Embedded System



Why contracts?

- Usually, CPS Software verification is *hard*!
- But *-*critical applications* need verification

Challenge 1: Quality requirements need to be formalized and proven, but

  - how to formalize them?
  - how to prove them?

Challenge 2: Proof can be computed in modules. If proof is ***modular*** it can be reused as a proof component in another proof

  - *Contracts serve this purpose*: they prove assertions about components and subsystems
  - Whenever an implementation of a component is exchanged for a new variant, the new must be proven to be **conformant** to the old contract. Then the old global proof still holds.

Rich Component Models of CPS

- Used for **component-based** software for CPS
- A ***rich component*** defines contracts in several *views* with regard to different *viewpoints*
  - A contract for functional behavior (*functional view*)
  - Several other quality contracts, e.g.,
    - Real-time behavior (*real-time view*)
    - Energy consumption (*energy view*)
    - Safety modes (*safety view*)
    - Movements (*dynamics view*)
- The **contract** (about the observable behavior) of a component can be described by *state machines* (***interface automata***) *or in logic* in each specific view
  - The interface automata encode infinite, regular path sets (traces)
  - They can be *intersected*, *unioned*, *composed*;
  - They are ***decidable*** and contracts can be ***proven***
- Instead of an automaton in a contract, temporal logic can be used as well and compiled to automata (**temporal logic contract**).

Assumptions about components' description

- A component has one thread of control
- A component is always in a finite set of (observable) states
- The behavior of a component can be described by a protocol automaton (interface automaton)
- The automaton states and transitions can be annotated in different views *multi-view automata* (MVA):
  - A ***real-time automaton*** - MVA with real-time annotations
  - A ***safety automaton*** - MVA with safety annotations
  - A ***dynamics automaton*** - MVA with dynamics equations (physical movement, electricity movement)
  - An ***energy automaton*** **-** MVA with energy consumption annotations

Quality Contracts for Components

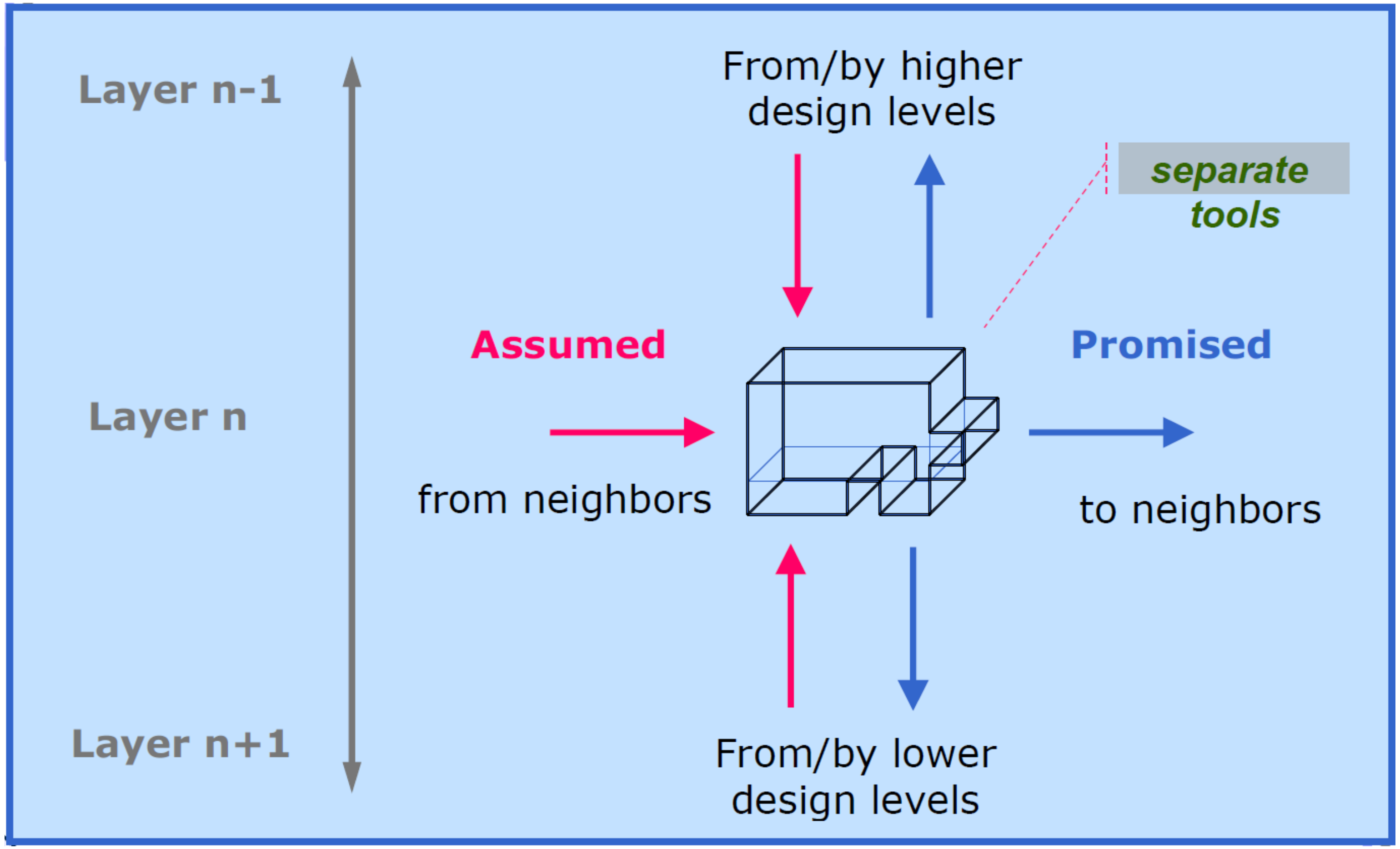
- ***Composability*** gives guarantees that a component property is preserved across composition/ integration
- ***Compositionality*** deduces global semantic properties of composed system from the properties of its components
- A ***contract*** is an *if-then rule*: under the assumption *A*, the component will deliver promise *P* (aka guarantee *G*)
- A ***quality contract*** is a contract in which view contracts form the assumptions and promises

Contract = (assumption, promise)

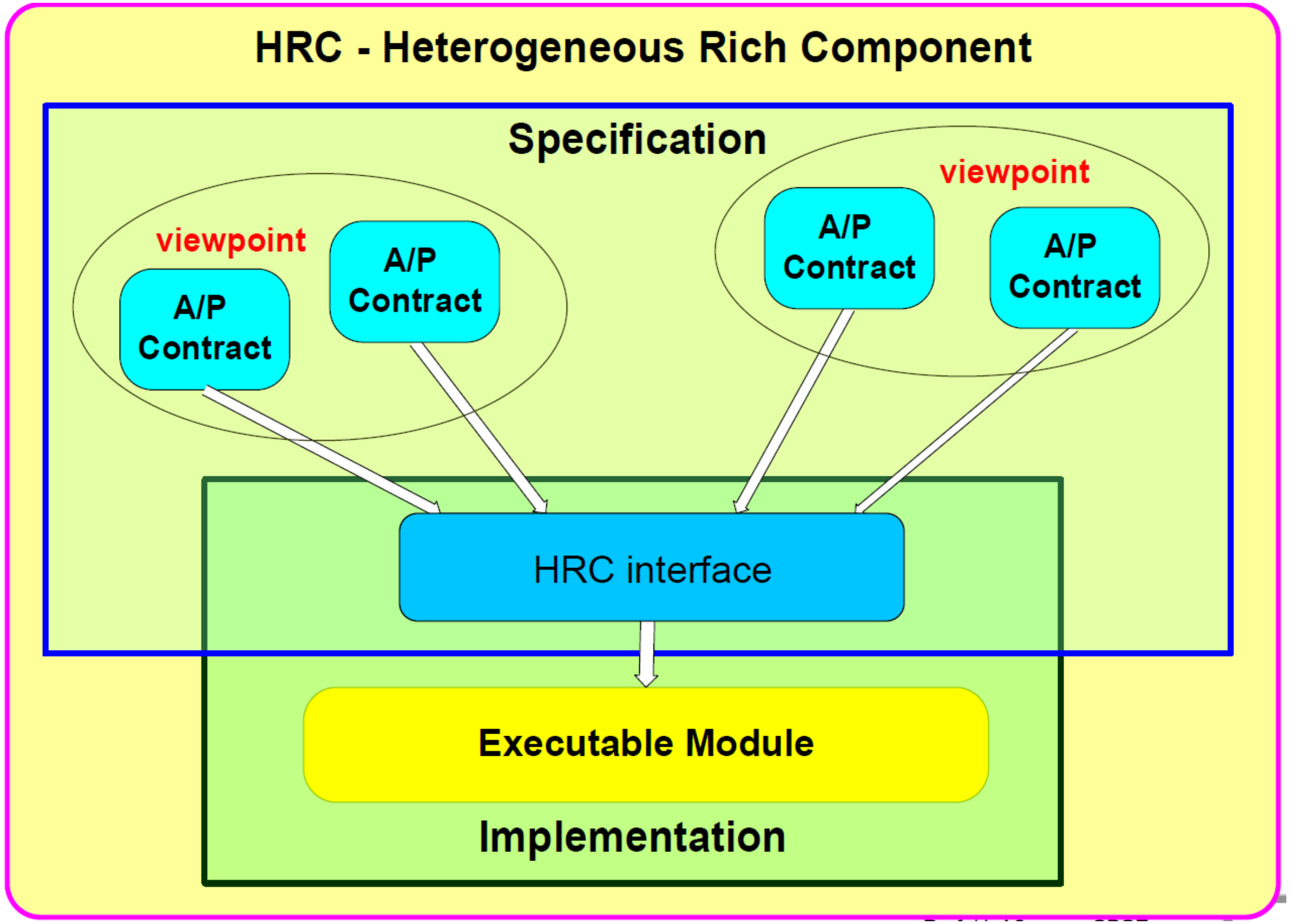
= IF assumption THEN promise

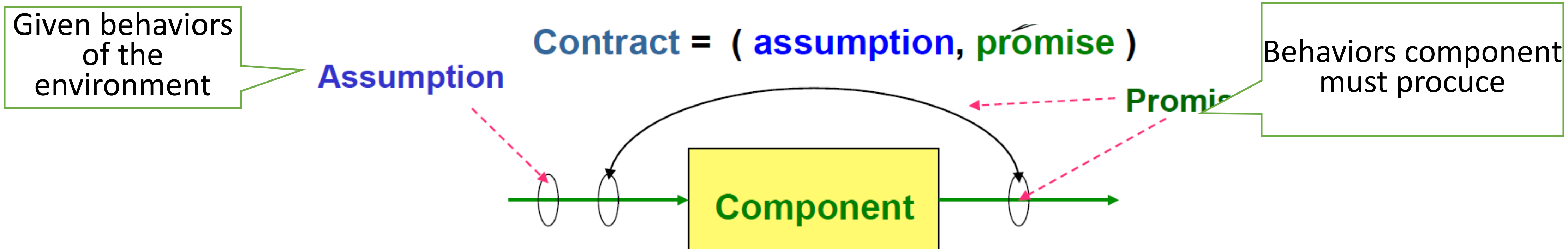
- **A/P-quality contract based component models must be *composable* and *compositional*.**

Speculative and Exploratory Design in Systems Engineering (EU SPEEDS Project)



Quality contract based component model





Assumption:

- Minimal delay between successive trains is 50 sec.

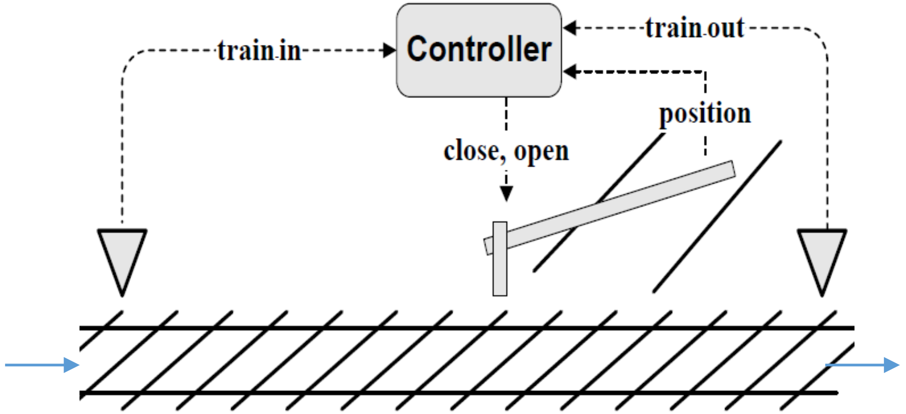
- At startup no train is already in crossing

- Trains move in one direction

Promise:

- Gate is closed as long as a train is in crossing

- Gate is open whenever XR is empty for more than 10 sec



Assertions Describe Behavior

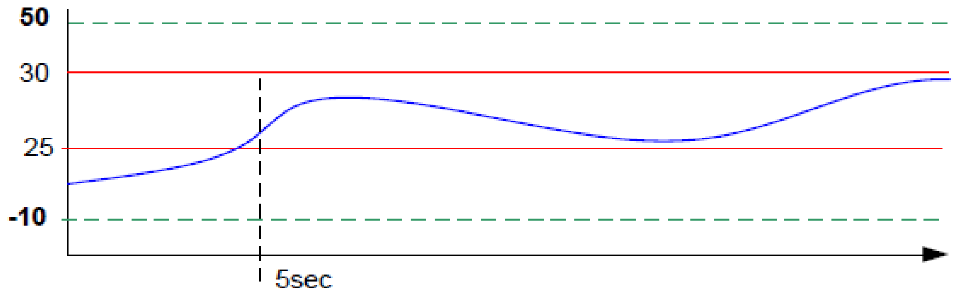
- An ***assertion*** specifies a subset of the possible component behaviors

  Contract **= (** ***assumption*****,** ***promise*** **)**

- ***Contract over continuous variable:***

  temp**: [-10°,50°]**

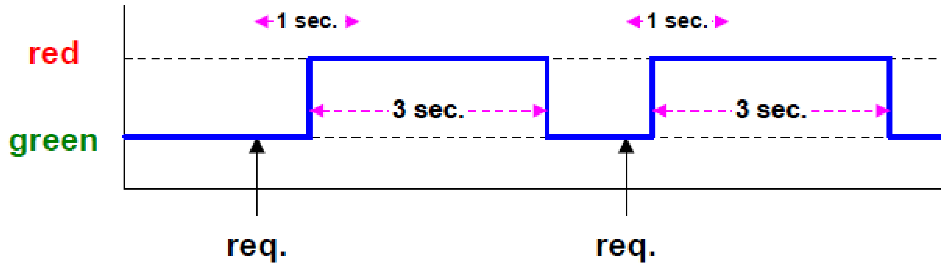
  'after 5 sec. 25 ≤temp ≤ 30'



- ***Contract over discrete variable:***

  lights **:{*****red*****,** ***green*****},** ***req*****:** ***event***

  'lights initially green, and after each 'req', within 1sec, become red for 3 sec. then back green'

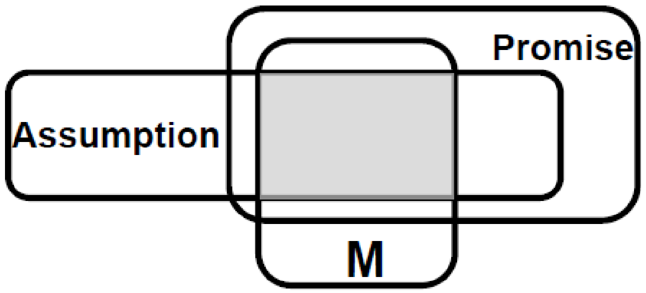


Basic Relations on Contracts

- *Satisfaction* relation (implementation conformance) couples implementations to contracts.
- Given contract: $C = (A, G)$, and implementation M
- **Satisfaction:** ***M*** **satisfies** ***C***

$$\boldsymbol{M \models C \;\Leftrightarrow_{def}\; A \cap M \subseteq G}$$

(promise *G* involves potentially more behaviors than the intersection of *A* and *M*)



Basic Relations on Contracts

Given 2 contracts: ***C* = (*A*, *G*)** and ***C' = (A', G')**, and implementation M*

- ***Dominance*: (*C dominates C'*) :**

  $C < C'$ iff $A' \subseteq A$ and $G \subseteq G'$ **%** *C assumes more and guarantees less than C'*

(*A* is weaker than *A'* and *G* is stronger than *G'*)

contravariant in A and G, i.e., when assumption A "expands", the promise G "shrinks";

***Example*:**

- C': A= *daylight* G= *video & IR picture*
- C: A'= *anytime* G'= *only IR picture*
- *Daylight* $\subseteq$ *anytime*, *video&IR picture* $\subseteq$ *IR picture*

***Claim*:** $M \models C$ and $C < C' \Rightarrow M \models C'$

(if *M* satisfies *C*, and *C* dominates *C'*, then *M* satisfies *C'*)

Compatibility of Contracts

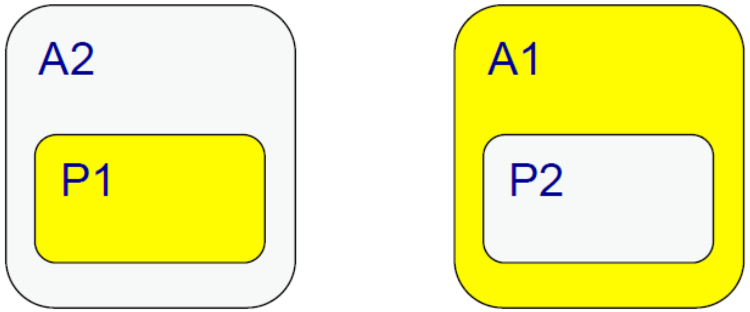
- Compatibility is a relation between two or more contracts *C*1 .. *Cn*
- Two contracts *C*1 and *C*2 are ***compatible*** whenever the promises of one guarantee that the assumptions of the other are satisfied
  - It means, when composing their implementations, the assumptions will not be violated and
  - the corresponding components "fit" together
- ***C*1 = (*A*1, *P*1) and *C*2 = (*A*2, *P*2) are compatible, denoted *C*1 <-> *C*2 iff**

$$P1 \subseteq A2 \text{ and } P2 \subseteq A1$$

*i.e. C*1 is compatible to *C*2 if *C*1.*P* is stronger (more restrictive) than *C*2.*A*, and *C*2.*P* stronger than *C*1.*A*

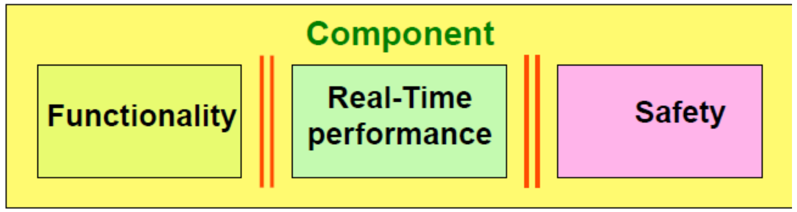
In logic:

$$P1 \Rightarrow A2 \text{ and } P2 \Rightarrow A1$$

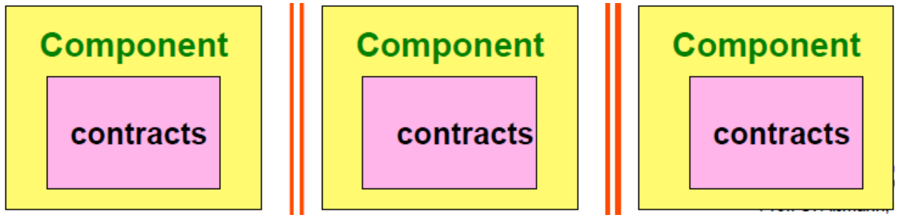


Composition of Contracts

- *within one component* (same interface), contracts in different views must be ***compliant, i.e. their conjunction is valid***:
  - The real-time assertions can be coupled with functional, safety, and energy view assertions



- *along components* – contracts of a certain viewpoint can be ***composed***



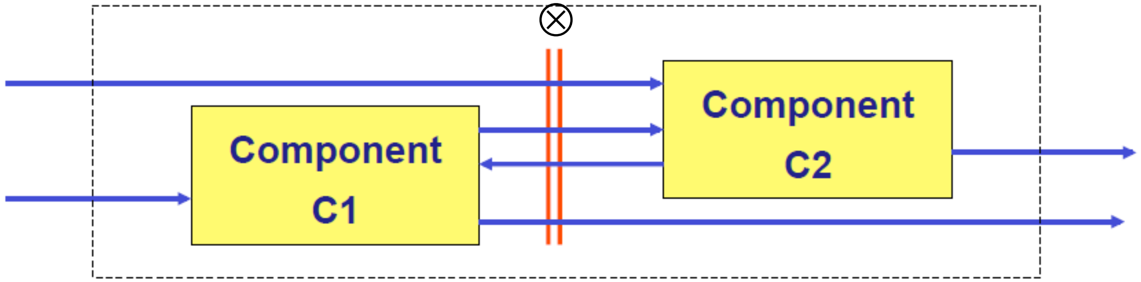
Contract operators: Parallel Composition of Contracts (of separate components)

- Given contracts $C1 = (A1, G1)$, $C2 = (A2, G2)$, and implementation M
- **Parallel composition** of contracts: $C1 \otimes C2 = (A, G)$

  where in set notation $A = (A1 \cap A2) \cup \neg(G1 \cap G2)$, $\quad G = G1 \cap G2$

  and in logic $A \Leftrightarrow (G1 \wedge G2) \Rightarrow A1 \wedge A2 \quad G \Leftrightarrow G1 \wedge G2$

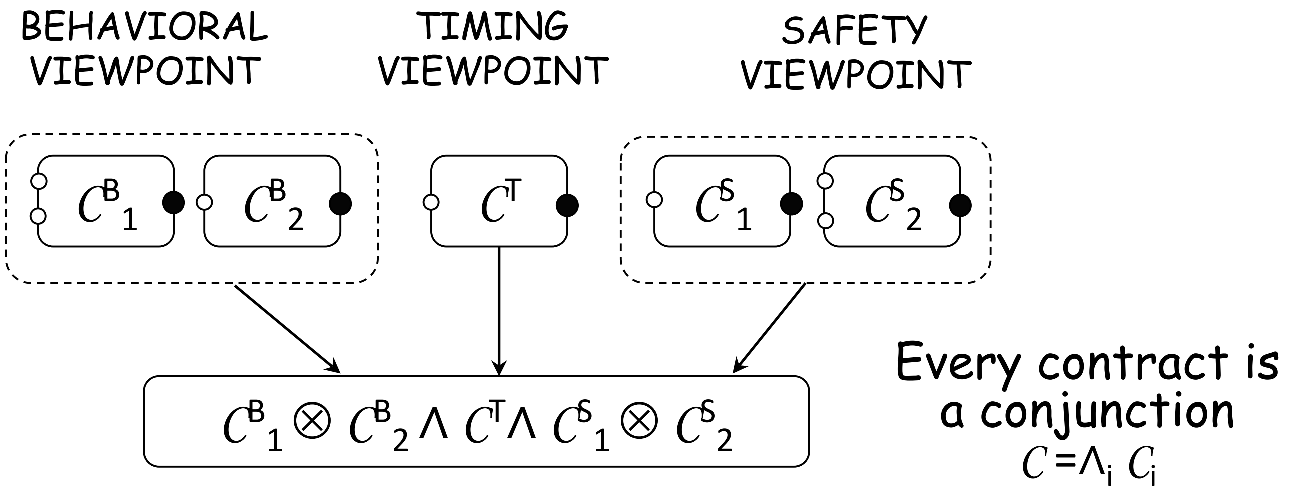
$$A = \max \left\{ A \;\middle|\; \begin{array}{c} A \wedge G2 \Rightarrow A1 \\ \textit{and} \\ A \wedge G1 \Rightarrow A2 \end{array} \right\}$$



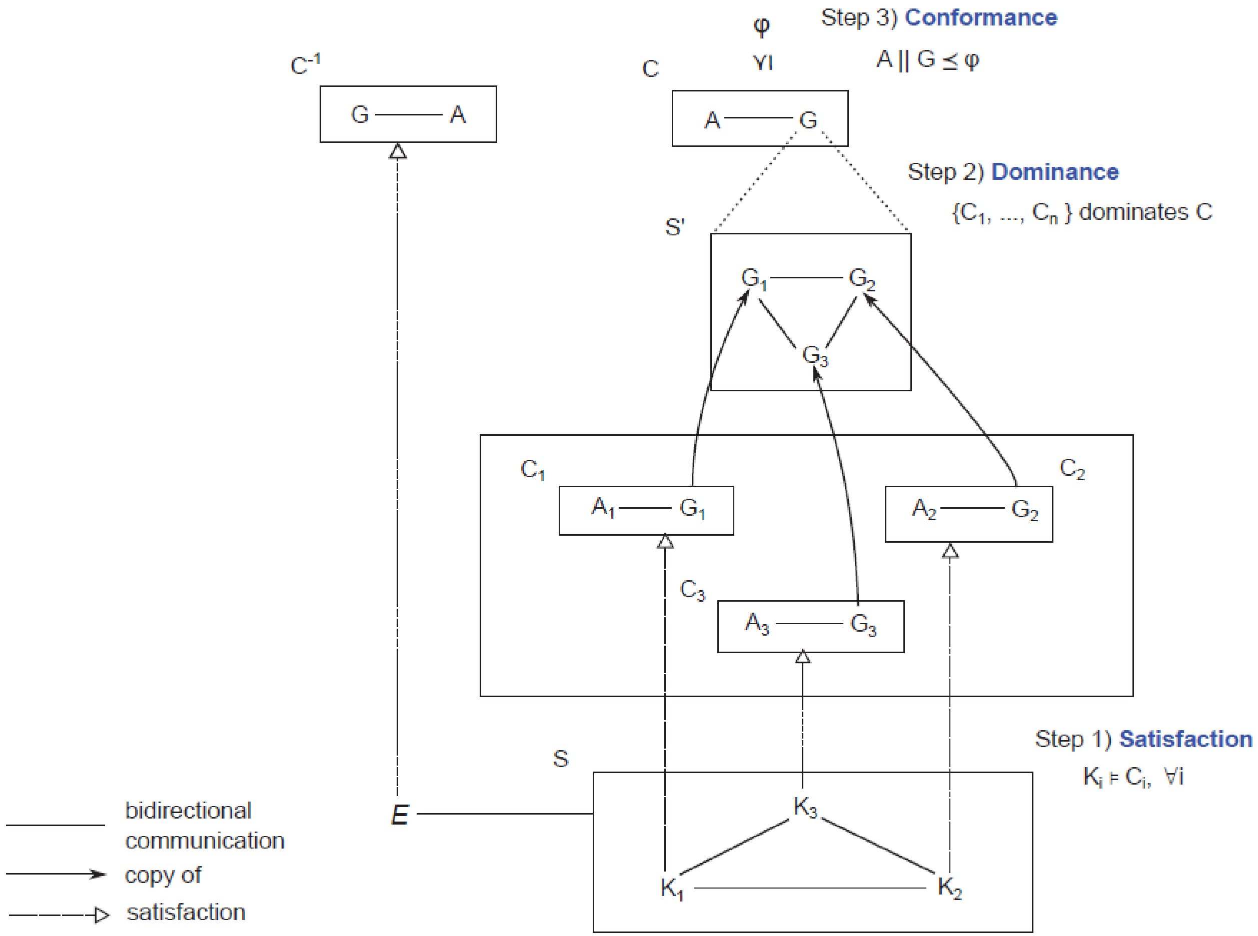
- where "max" refers to the order of predicates by implication;
- A is the weakest assumption such that the two referred implications hold.

Contract Operators: Conjunction (∧) (viewpoint fusion)

- Supports separation of different design concerns;
- Contract can be a conjunction of multiple viewpoints, each covering a specific concern of the design and specified by an individual contract $\mathcal{C}$.



Example of contract-based reasoning: a three-component subsystem $K_1 \parallel K_2 \parallel K_3$



Next lecture

- How to annotate component programs??
- How to verify satisfiability of contracts
- Hoare logic for Key tool
- How to test conformance

Announcement

- More about *contract engineering* and formal verification of contract based systems can be learned in Formal Methods Course ITI0130/ ITI8530 , Spring 2018.