Contracts and Interfaces in the context of Requirements Engineering

July 2012, ICECSS

Albert Benveniste

INRIA Rennes

This talk is the result of a collective effort of a large community

- B. Caillaud, J-B Raclet, A.Benveniste
- R. Passerone, A. Sangiovanni-Vincentelli
- T. Henzinger, D. Nickovic
- K. Larsen, W. Damm

Contents

Industrial context

- OEM/supplier chains, concurrent development
- Virtual engineering
- Stating responsibilities, contracts

OEM/supplier chain: some figures regarding a real example in aeronautics

Navigation system

- a “quite-complex” sub-system developed by a tier-1 sub-contractor for a larger system
- still, ≈ 3000 requirements

Approximately 200 engineers in-house

Major sub-contractors ~10, 100 engineers

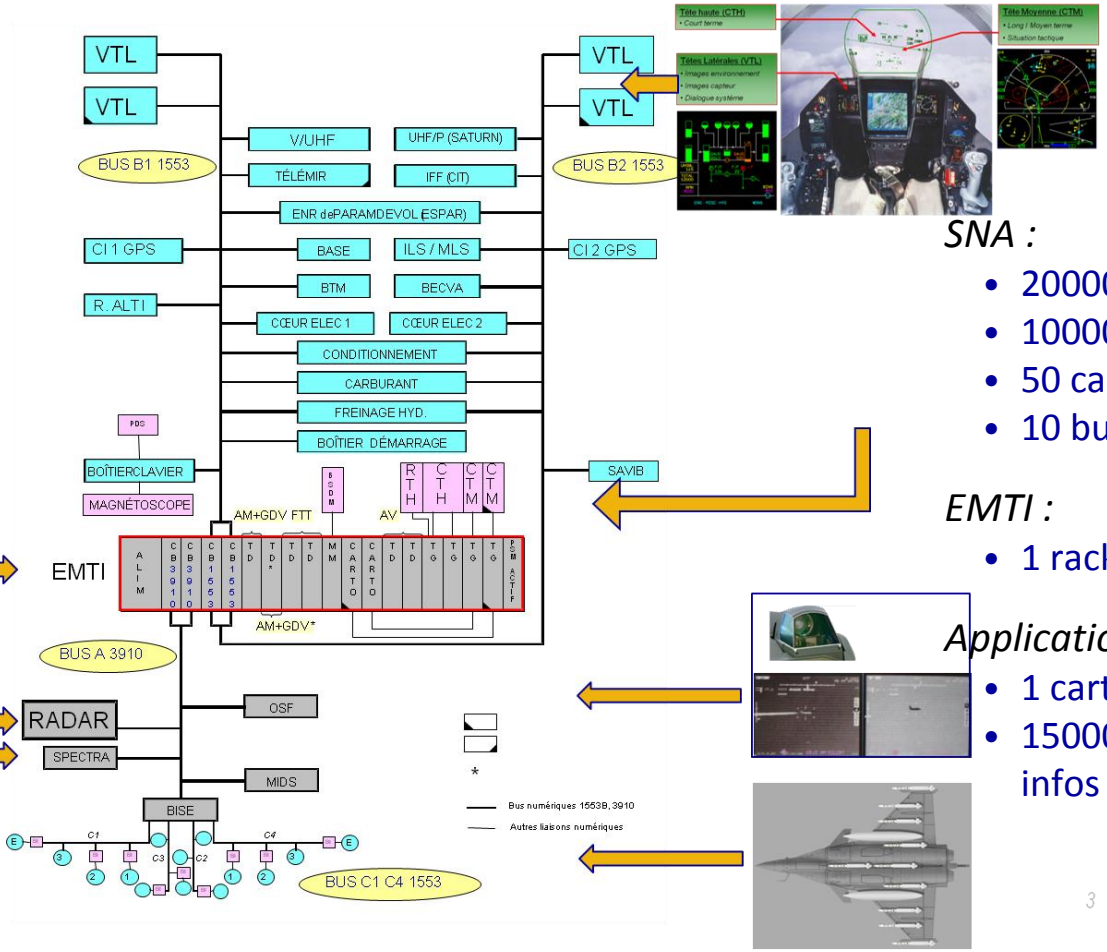
- For SW development and V&V
- Each contract in the magnitude order of 1M€

Système de Navigation et d'Attaque Rafale : *Architecture logique*

Architecture interne :
- Calculateurs
- Réseaux de données
- Logiciels

Architecture externe :
- Capteurs
- Communications
- Guerre électronique
- Armements

Architecture cockpit :
- Relation Homme système
- Commandes et visualisations

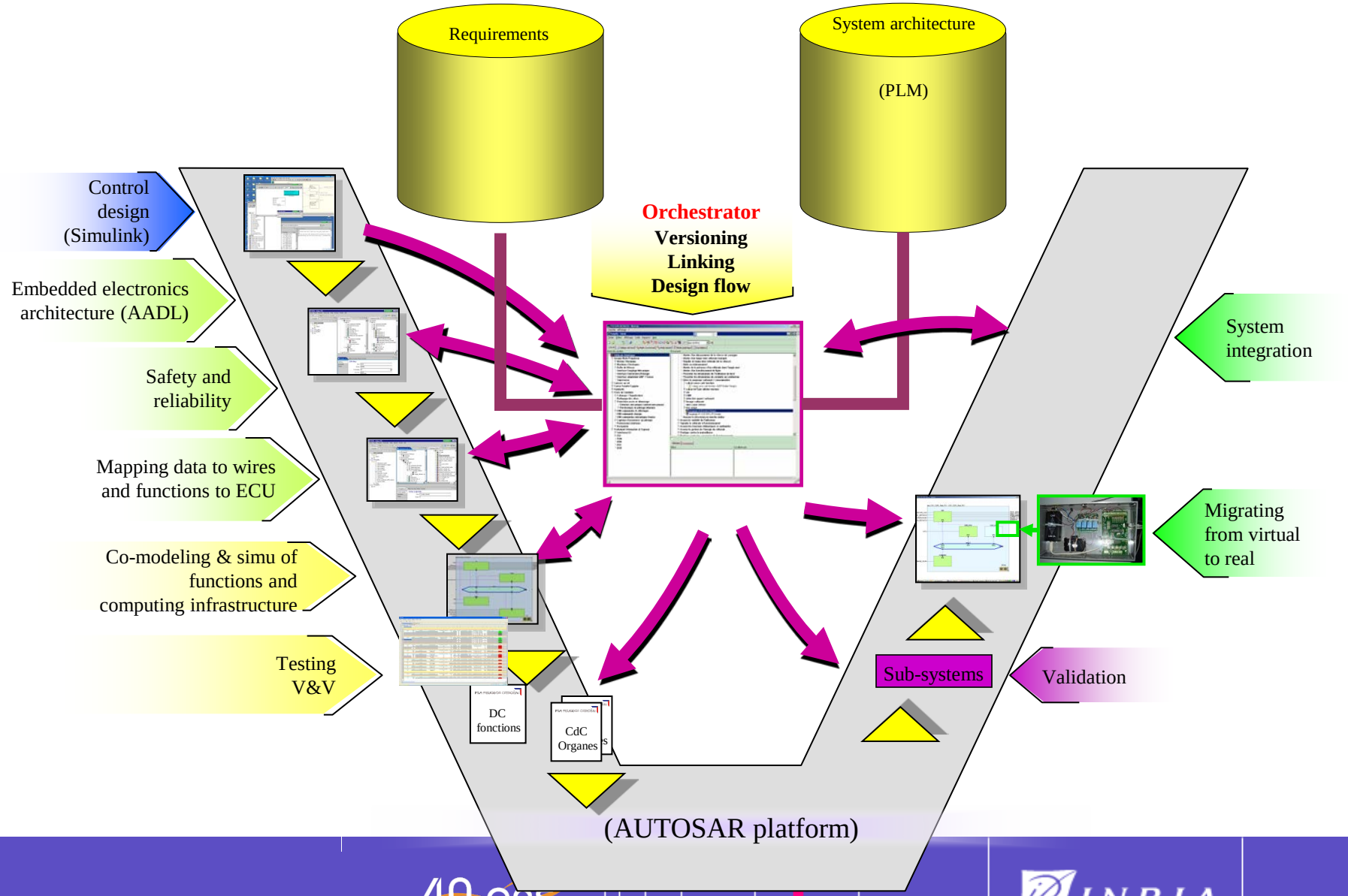


SNA :
- 200000 exigences
- 100000 infos numériques
- 50 calculateurs - 100 boîtes noires
- 10 bus numériques, 100 coupleurs

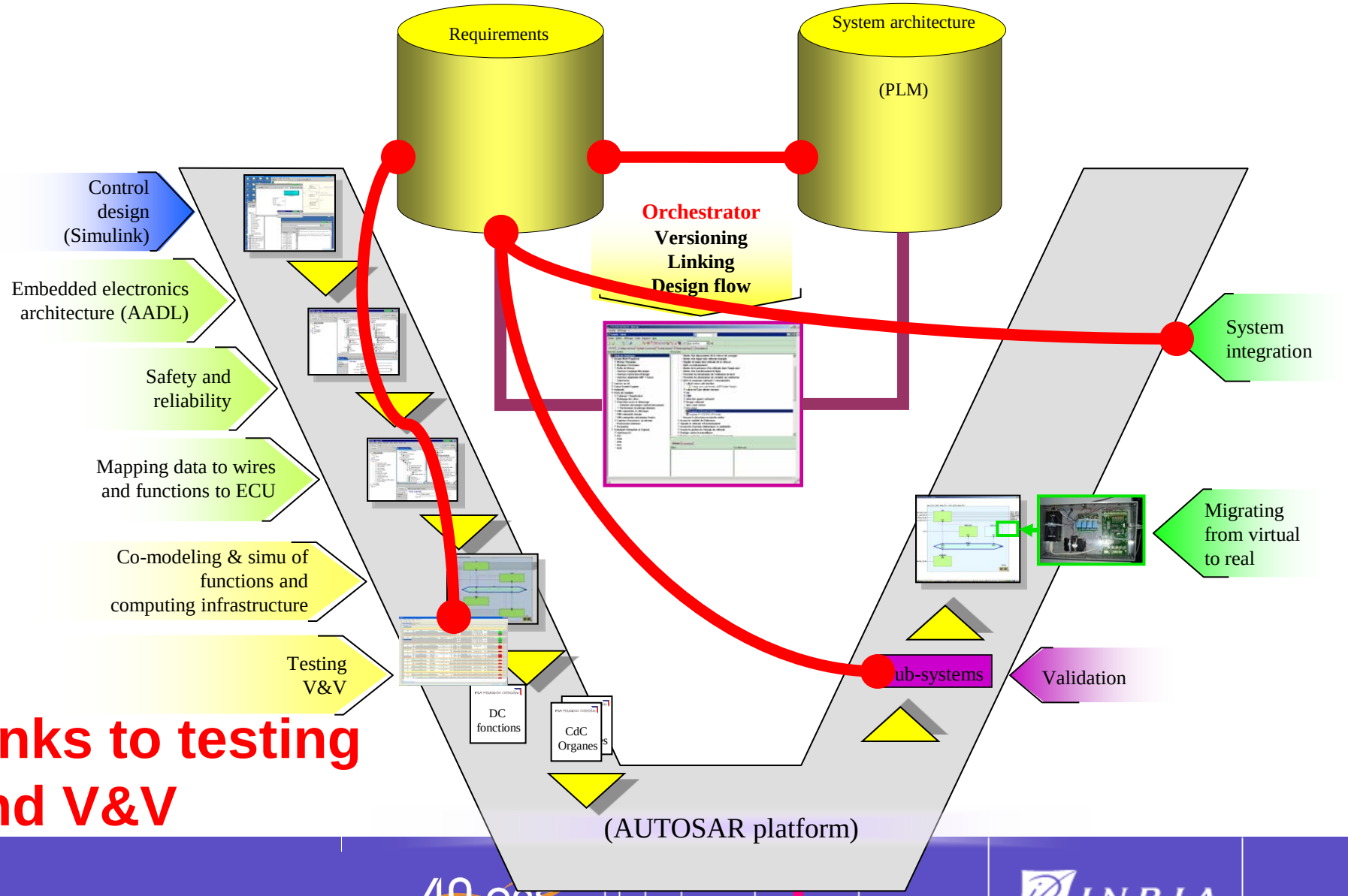
EMTI :
- 1 rack, 18 cartes

Application Mission :
- 1 carte, 7 tâches temps réel
- 15000 infos applicatives, 40000 infos numériques

A sample tool chain



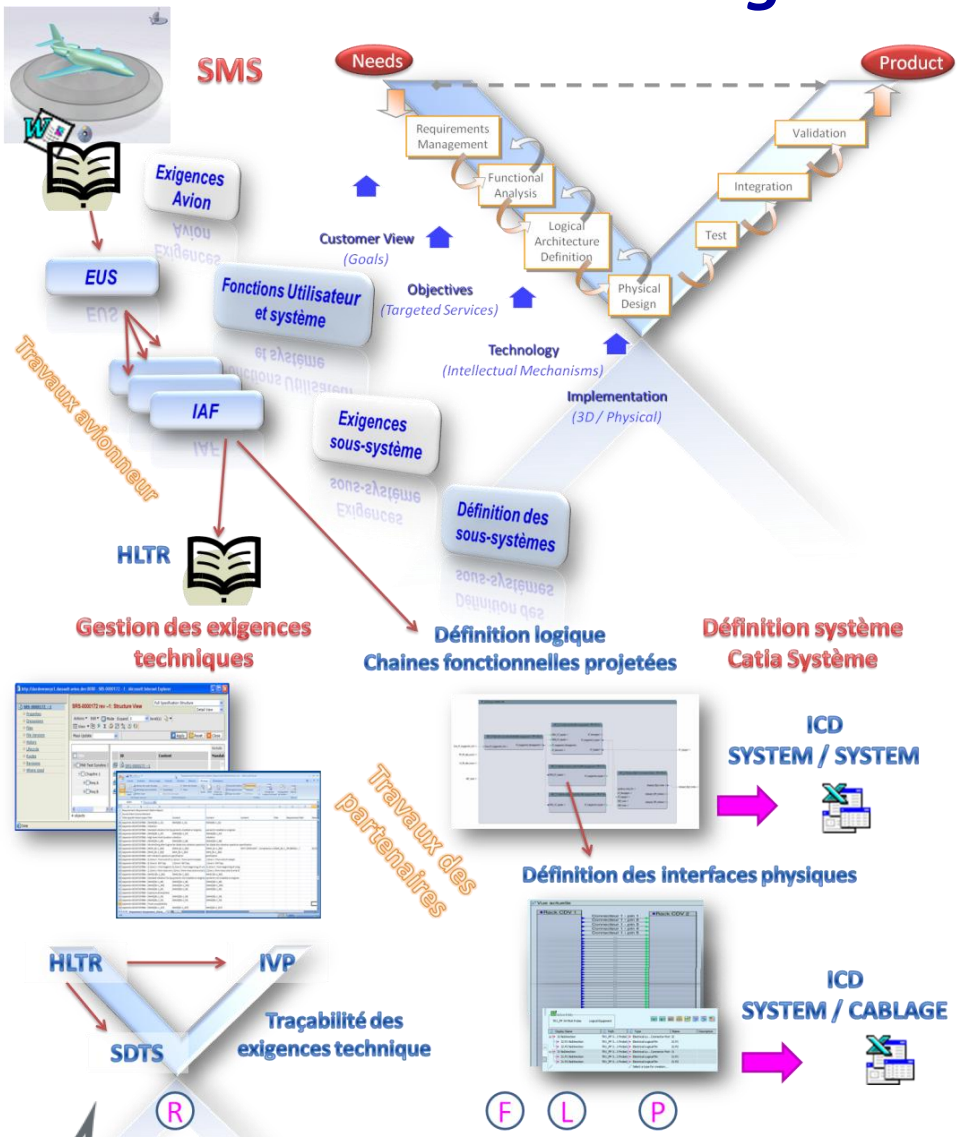
A sample tool chain: links to requirements

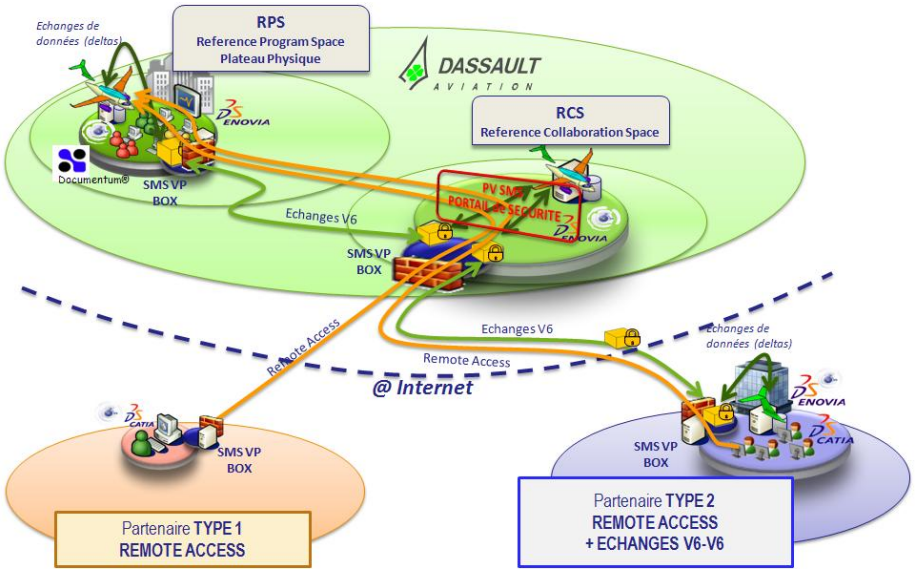


Links to testing and V&V

Processus outillé d'ingénierie système :

PLM Système
et plateau virtuel





opérationnels
sur le programme SMS

Real examples of requirements

Requirements documents are structured into **viewpoints**

(sometimes referred to as *chapters, aspects, sub-documents,* or whathaveyou, depending on the company or sector)

Real examples of requirements
The monitoring viewpoint

Software for monitoring the physical system

- Requirements must specify
  - Decision logic
  - Sampling period
  - When to activate/inhibit monitoring
  - Time lag before fault confirmation
  - Fault identifier
  - Fault compensation
  - Returns to the cockpit
  - *Very diverse in nature*
- Requirements must also specify *at a higher specification level* what is the objective of the detection, which fault to detect and isolate
  decision logic ≈ implementation

Real examples of requirements from the monitoring viewpoint

Monitoring of	Test logic	Conf	Period	Test mode / act. phase	Sanction name	Treatment before conf	inhib. by	Environment data
RDC CAN Bus monitoring (Lost data during 3s)	if GNDTST_DATALOADSW is not refreshed	3s	1s	CBIT (permanent)	Use default value for GNDTST_DATALOADSW	Use the last valid GNDTST_DATALOADSW data	-	Brake_Intern_Context
Breaker	If during 40 AS frames [the status B_BREAKER_STAT4_IN is False (= Breaker4 opened) and B_WATCHDOG_DSP4_IN is False] 14 times or more	-	5ms	CBIT : (100ms after B_INV4_INHIB_OUT is set to TRUE) AND BOT_STATUS_IN different from 0x0F AND IT_STATUS_IN different from 0x0F	INHIB_EBA4	-	-	EBA4_Intern_Context

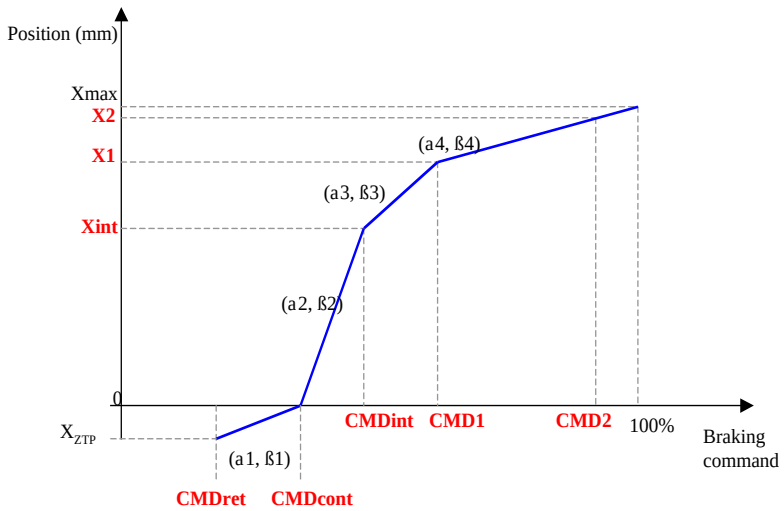
Real examples of requirements from the functional viewpoint (1)

When the Engine Run Park mode is controlled **and** the park status information is FALSE and park_release_is_running is FALSE, the XXX shall :

- control each YYY to hold its position and energize all friction brakes, until the friction brakes are unlocked.
- **then**, apply a braking command with a braking force reference (F_BR) equal to C_ERP_CMD, until all valid YYYs have reached the braking command (braking_command_reached).
- **then**, command each YYY to hold its position and de-energize all friction brakes, until the friction brakes are locked.
- **then**, command each YYY to switch to the YYY Standby state.
- **AND** set B_POWERREQUEST_OUT to false**.**

Seems quite low-level…

Real examples of requirements from the functional viewpoint (2)



Req_1: The EBA Position command shall be Position = a1 * F_BC + b1 when F_BC is lower than or equal to CMDcont

Req_2: The EBA Position command shall be Position = a2 * F_BC + b2 when F_BC is higher than CMDcont and lower than or equal to CMDint

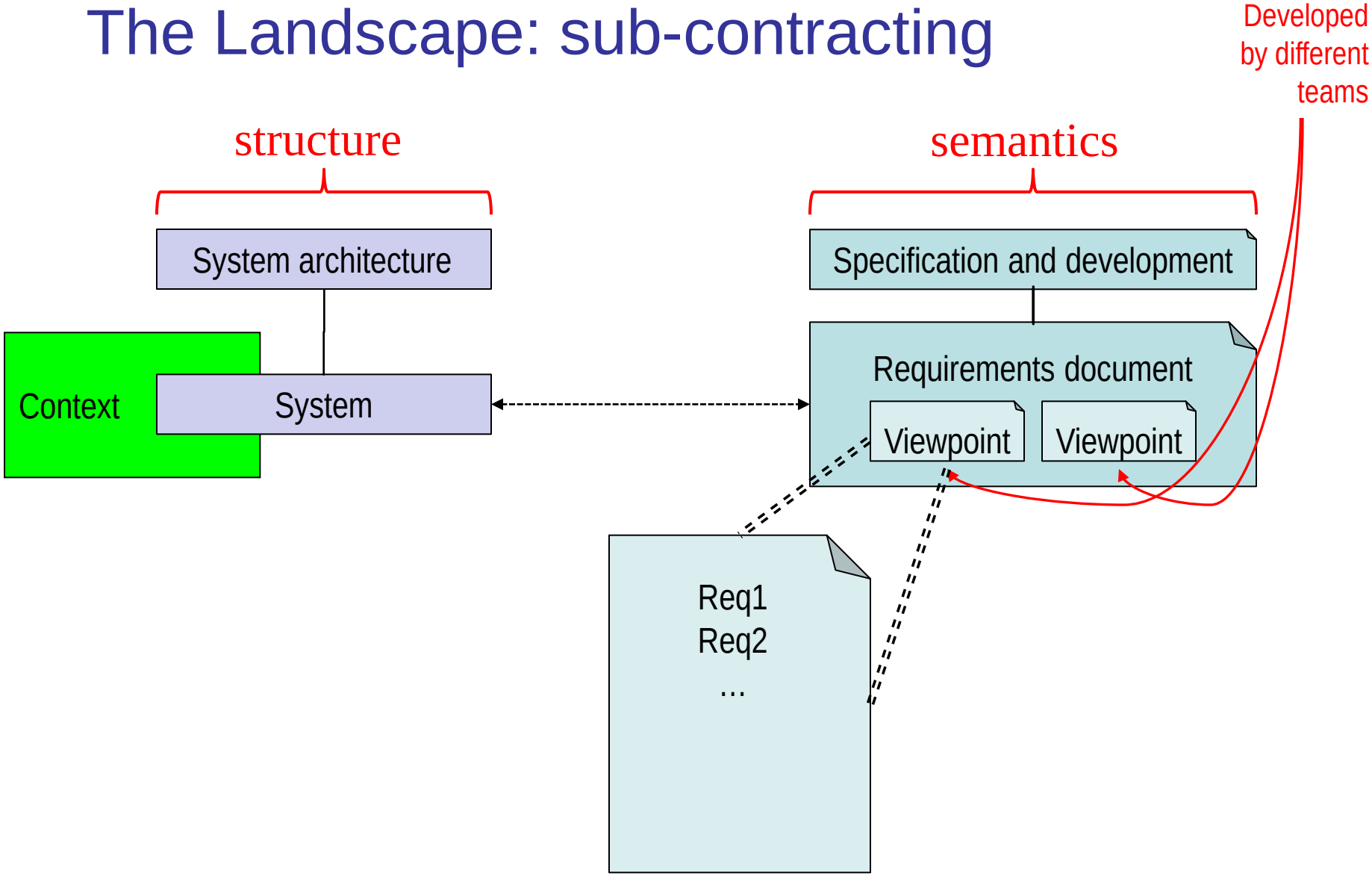
Req_3: The EBA Position command shall be Position = a3 * F_BC + b3 when F_BC is higher than CMDint and lower than or equal to CMD1

Req_4: The EBA Position command shall be Position = a4 * F_BC + b4 when F_BC is higher than CMD1

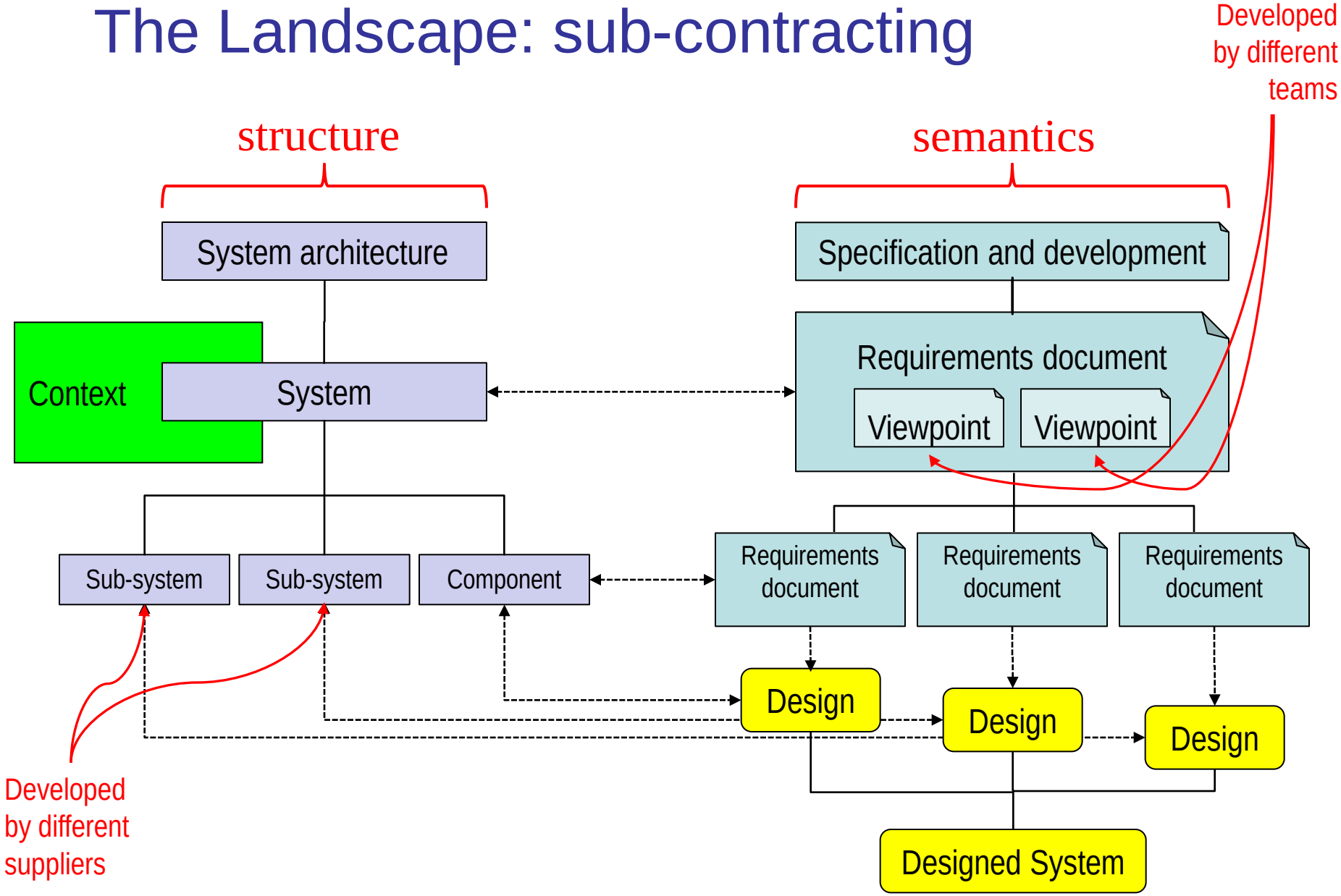
Requirements on requirements engineering

- Landscape: system architecture, requirements, design
- Ontology
- Exploring requirements
- Partitioning and sub-contracting
- Modular handling of Viewpoints & Subsystems
- Fundamental properties

The Landscape: sub-contracting



The Landscape: sub-contracting



Requirements on requirements engineering

Traceability

- Requirements attached to “everything” via hyperlinks (tests, V&V, integration)

Ontology

- Terms used for entities should be precise and unambiguous (important)
- Terms used for entities should be structured (ontology)

Identifying responsibilities

- Some requirements express guarantees; other express assumptions

Partitioning and sub-contracting

- Allocating requirements to suppliers, budgeting

Modular handling of viewpoints & subsystems

- Separation of concerns: function, QoS, safety/reliability…

Fundamental properties (certification bodies)

- Completeness, Consistency, Compatibility, … (from INCOSE)

Requirements on requirements engineering

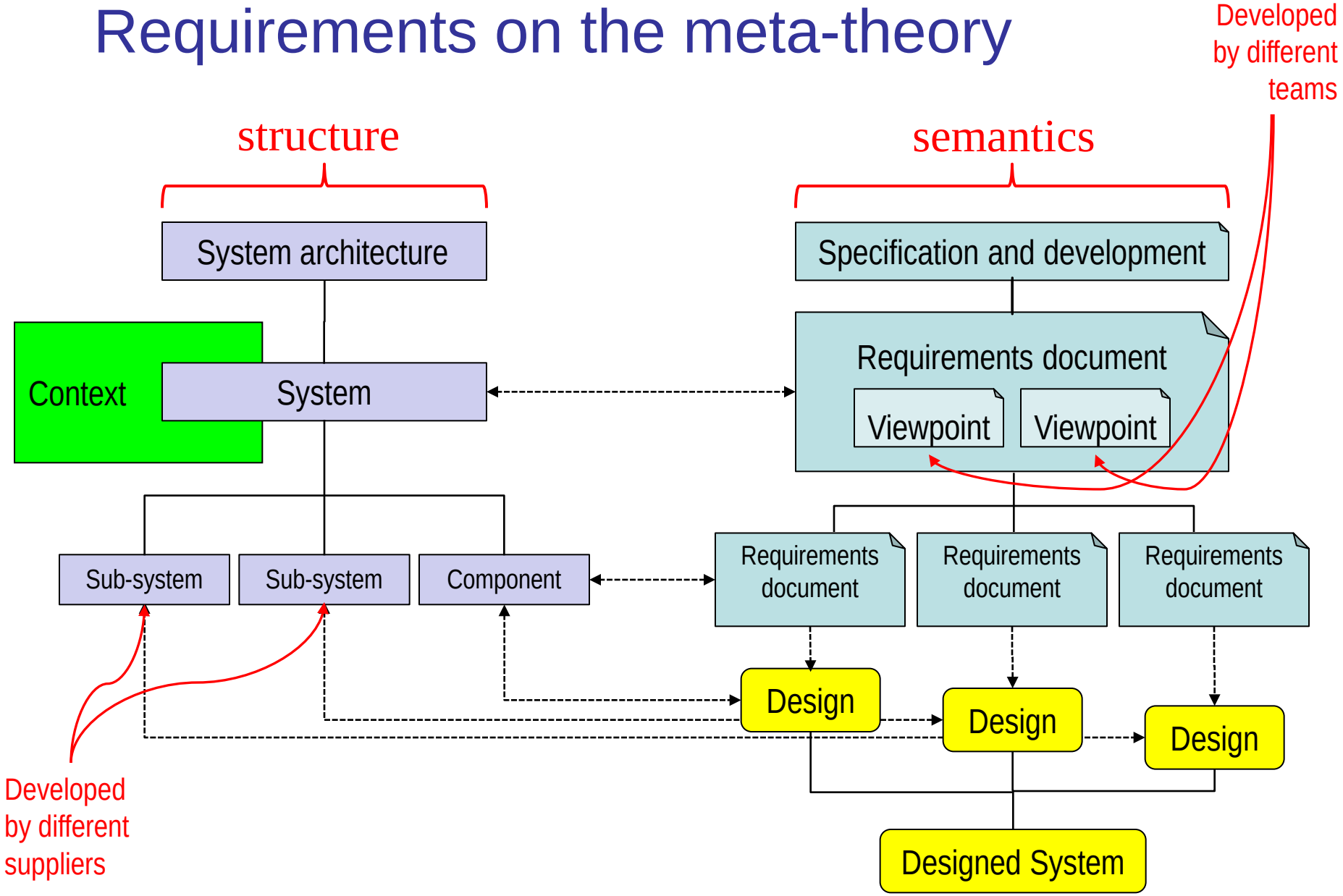
Overall, requirements engineering

- has been considered by the AI community (ontologies)
- has been considered by the Software Engineering community as part of MDE
- has been mostly ignored by other research communities
  - control science
  - formal methods in computer science

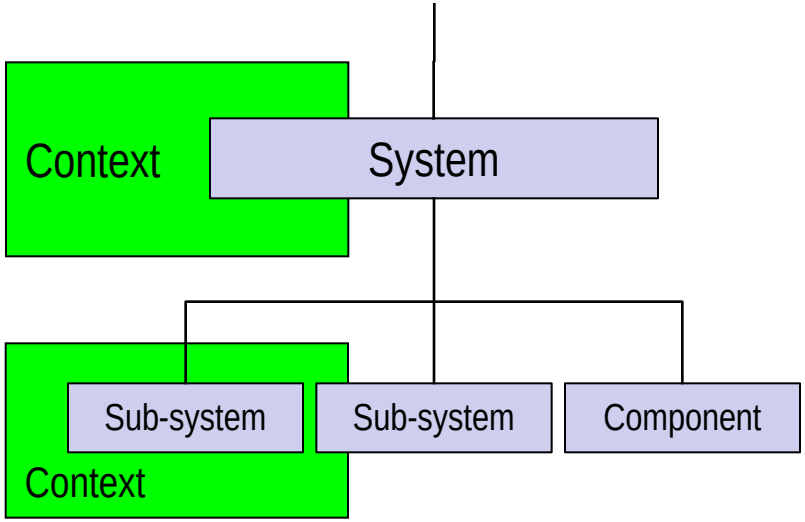
Requirements by Contracts

A meta-theory of contracts

Requirements on the meta-theory



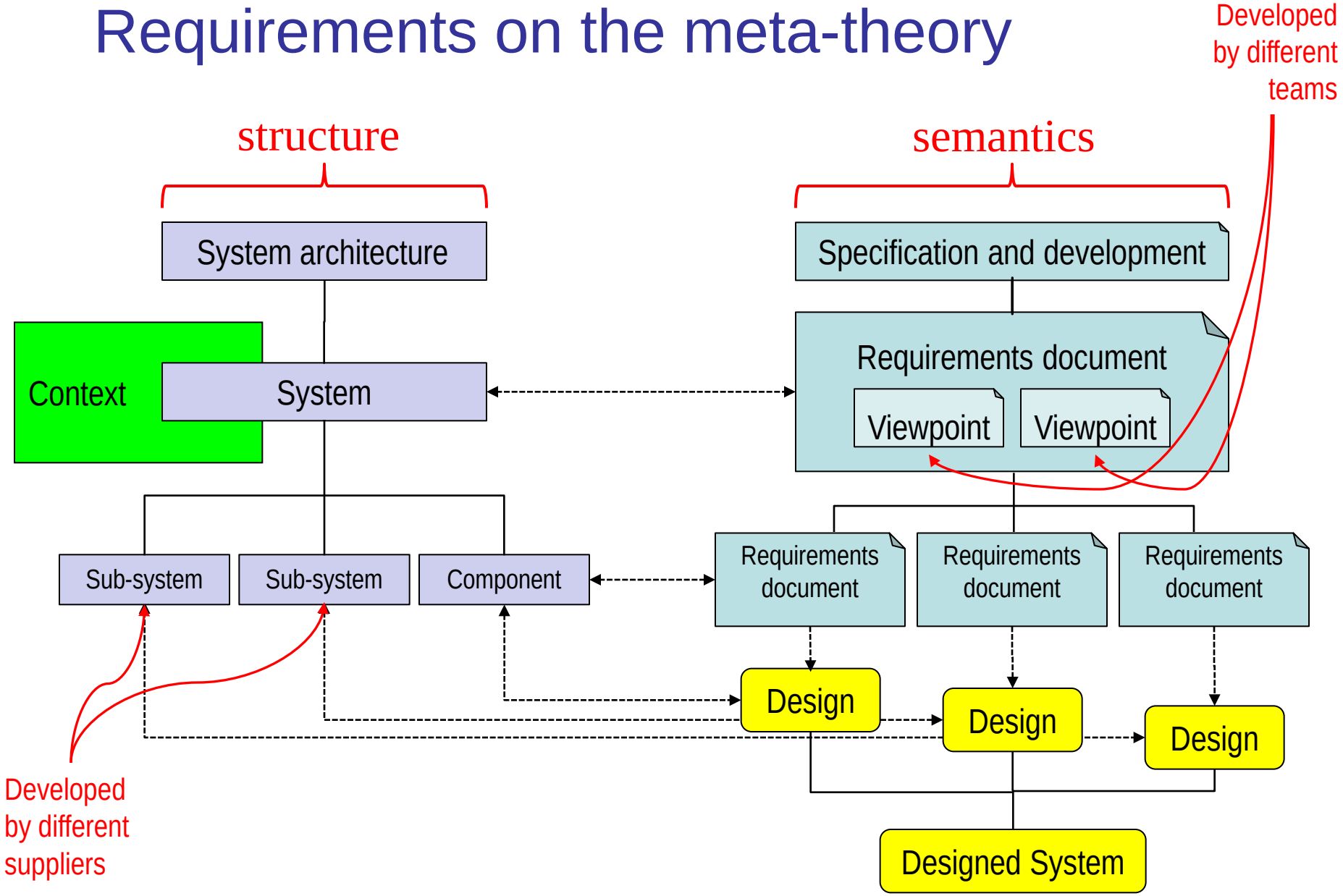
Requirements on the meta-theory {environment, component}



Contexts are important

- What the system guarantees: must be met by any implementation
- What the system assumes about its context of use: must be met by any legal environment

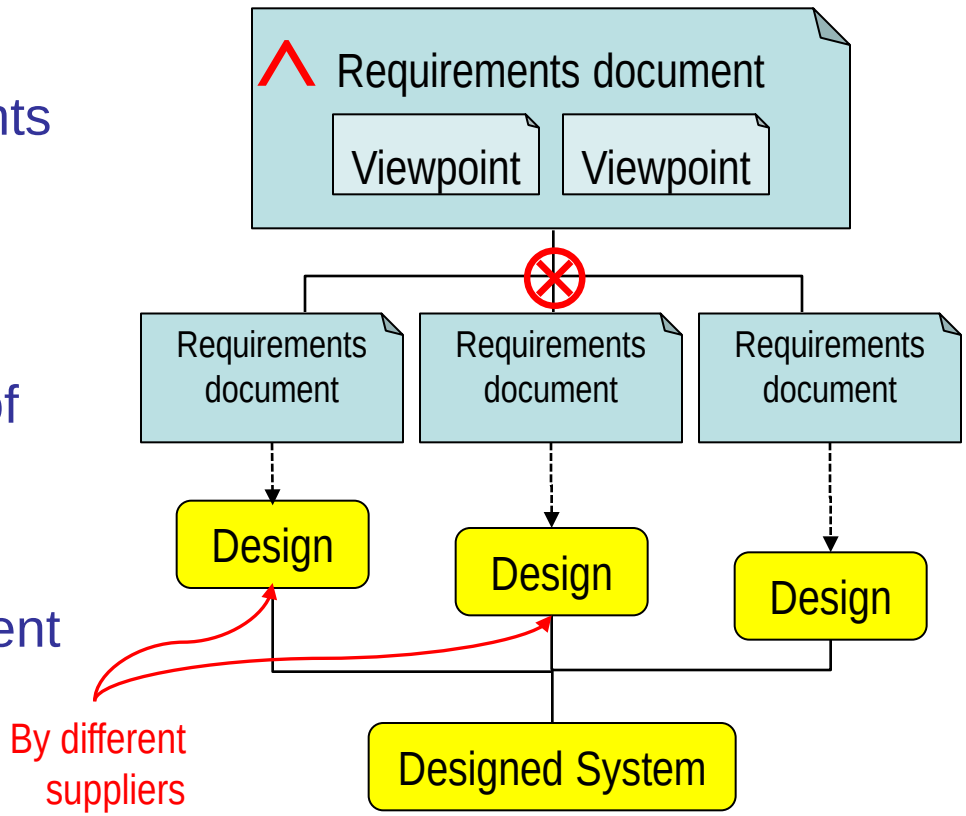
Requirements on the meta-theory



Requirements on the meta-theory
conjunction and parallel composition

From $\wedge$ to $\otimes$

- Requirements documents decompose into chapters/viewpoints: conjunction $\wedge$
- System = architecture of sub-systems: composition $\otimes$
- Independent development



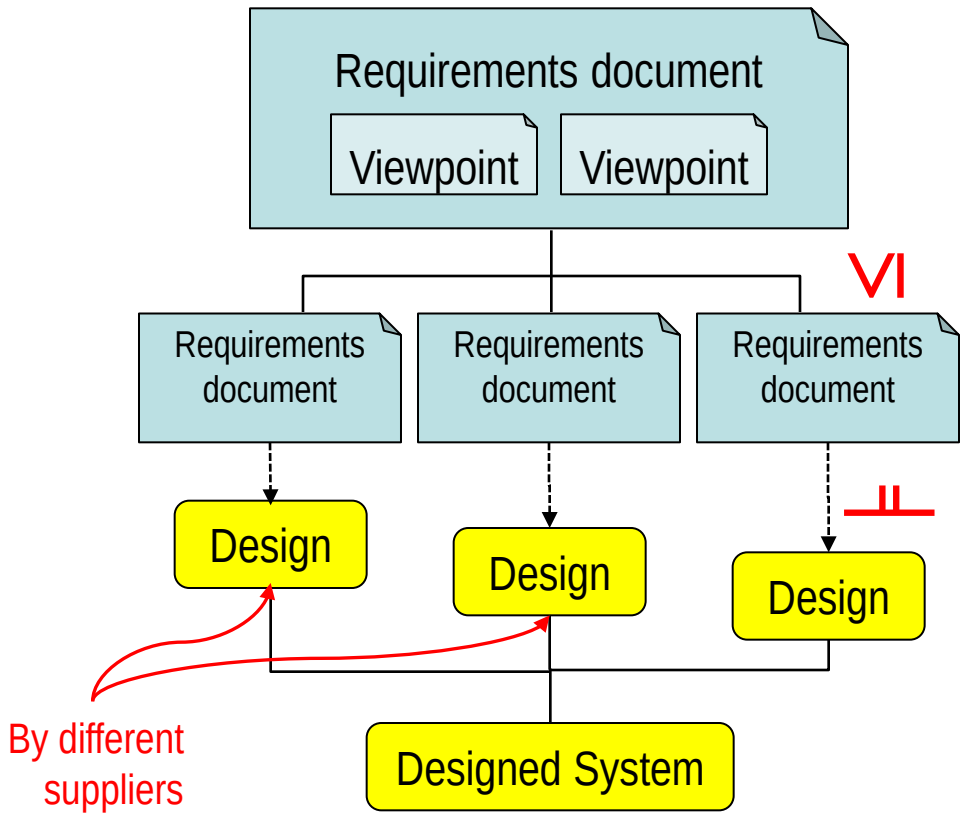
Requirements on the meta-theory implements and refines

Designed component
$\models$ local contract

- Meets the guarantees under any legal environment

Decomposed contract
$\leq$ global contract

- Stronger guarantees
- Relaxed context



Let’s go for the maths…

The meta-theory

- We assume some primitive concepts

Component	M *open* or *closed*
Composability	A type property
Composition	$\times$ commutative & associative
Environment	$E : E \times M$ closed

- On top of these primitive concepts we define generic operators satisfying generic properties
- How primitive concepts, operators, and properties, are made effective depends on the specific framework

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$
Composition	$\mathcal{C}_1 \otimes \mathcal{C}_2 = \bigwedge \left\{ \mathcal{C} \;\middle\vert\; \begin{array}{c} [M_1 \models^{\mathsf{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathsf{M}} \mathcal{C}_2] \Rightarrow M_1 \times M_2 \models^{\mathsf{M}} \mathcal{C} \\ \text{and} \\ E \models^{\mathsf{E}} \mathcal{C} \Rightarrow [E \times M_2 \models^{\mathsf{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathsf{E}} \mathcal{C}_2] \end{array} \right\}$

The meta-theory

- Generic Relations and Operators:

Contract	$\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c) =$ (set of environments , set of implementations)
Consistency	$\mathcal{M}_c \neq \emptyset$
Compatibility	$\mathcal{E}_c \neq \emptyset$
Implementation	$M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_c$; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_c$
Refinement	$\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}_c$ and $\mathcal{M}_{c'} \subseteq \mathcal{M}_c$
Conjunction	$\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$
Composition	$\mathcal{C}_1 \otimes \mathcal{C}_2 = \bigwedge \left\{ \mathcal{C} \;\middle\vert\; \begin{array}{c} [M_1 \models^{\mathsf{M}} \mathcal{C}_1 \text{ and } M_2 \models^{\mathsf{M}} \mathcal{C}_2] \Rightarrow M_1 \times M_2 \models^{\mathsf{M}} \mathcal{C} \\ \text{and} \\ E \models^{\mathsf{E}} \mathcal{C} \Rightarrow [E \times M_2 \models^{\mathsf{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathsf{E}} \mathcal{C}_2] \end{array} \right\}$
Quotient	$\mathcal{C}_1 / \mathcal{C}_2 = \max\{ \mathcal{C} \mid \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1 \}$

The meta-theory

- Generic Properties:

Refinement	substituability $\nearrow$ of environments substituability $\searrow$ of implementations
Composition	$\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$ independent implementability $\underbrace{(\mathcal{C}_1 \otimes \mathcal{C}_2) \otimes (\mathcal{C}_3 \otimes \mathcal{C}_4) = (\mathcal{C}_1 \otimes \mathcal{C}_3) \otimes (\mathcal{C}_2 \otimes \mathcal{C}_4)}_{\text{compatibility} \quad \Longrightarrow \quad \text{compatibility}}$ associativity $[(\mathcal{C}_{11} \wedge \mathcal{C}_{21}) \otimes (\mathcal{C}_{12} \wedge \mathcal{C}_{22})] \preceq [(\mathcal{C}_{11} \otimes \mathcal{C}_{12}) \wedge (\mathcal{C}_{21} \otimes \mathcal{C}_{22})]$ distributivity (freedom in design processes)
Quotient	$\mathcal{C} \preceq \mathcal{C}_1/\mathcal{C}_2 \;\Leftrightarrow\; \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1$

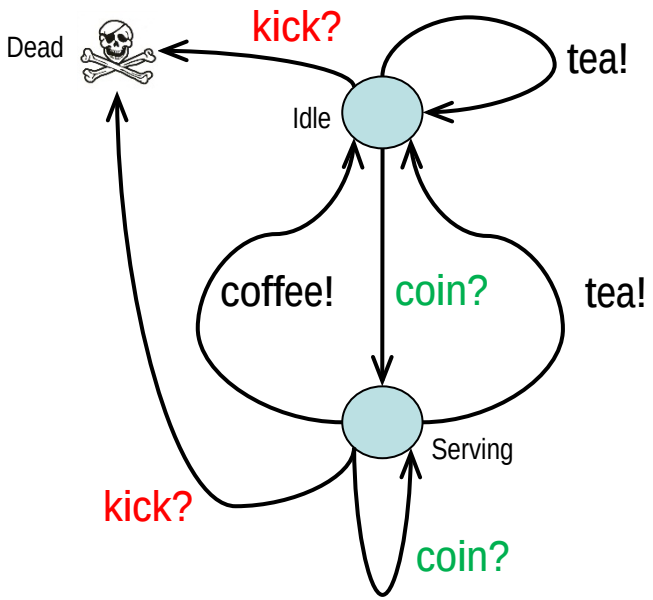
Concrete instances of the meta-theory

Concrete instances of the meta-theory

Component model

Input-output Automata

[Lynch]

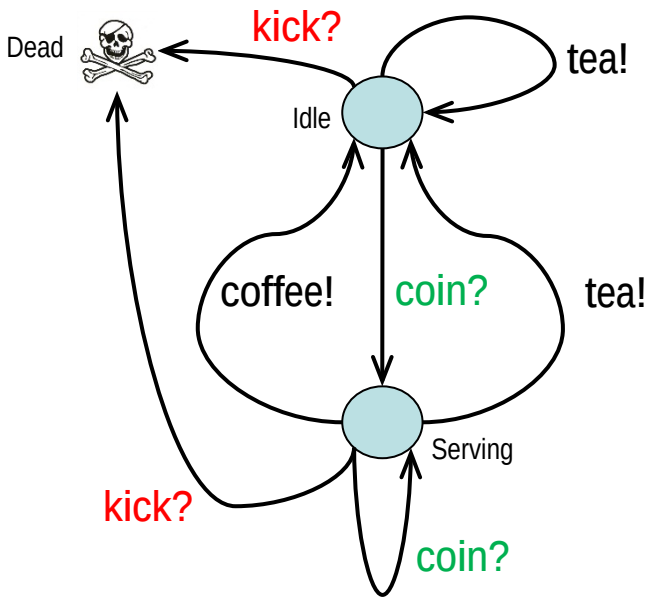


Concrete instances of the meta-theory

Component model

Input-output Automata

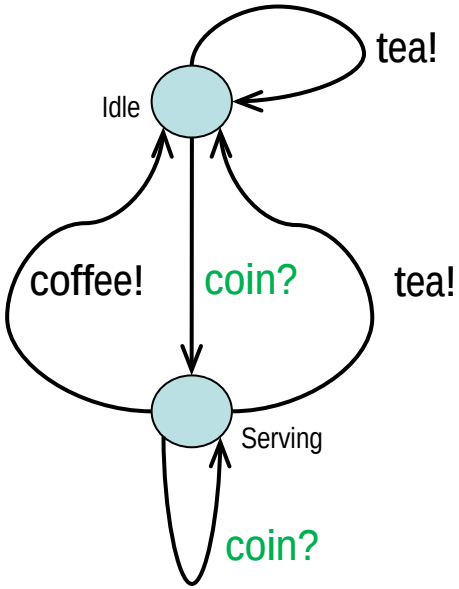
[Lynch]



Contract model

Interface Automata

[de Alfaro Henzinger]

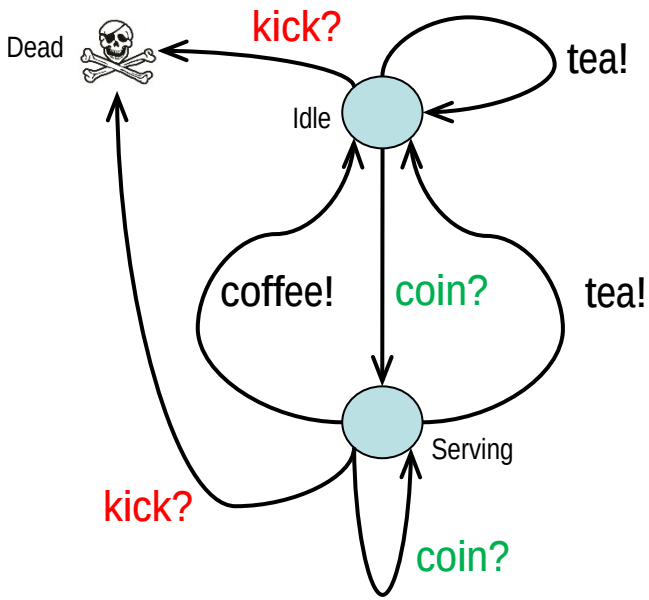


Concrete instances of the meta-theory

Component model

Input-output Automata

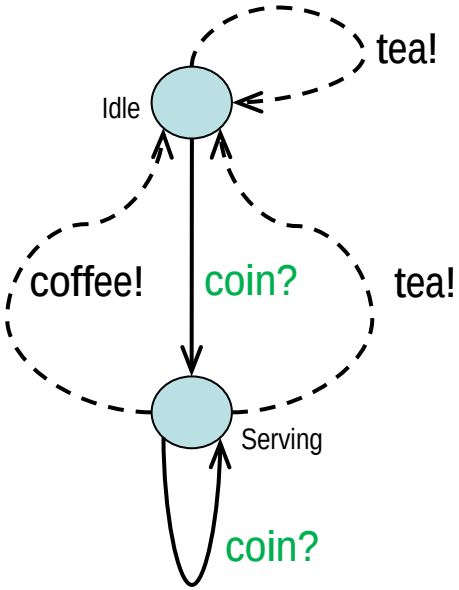
[Lynch]

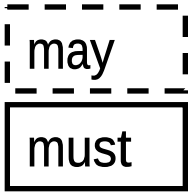


Contract model

Modal Interfaces

[Larsen & al] [Raclet & al]



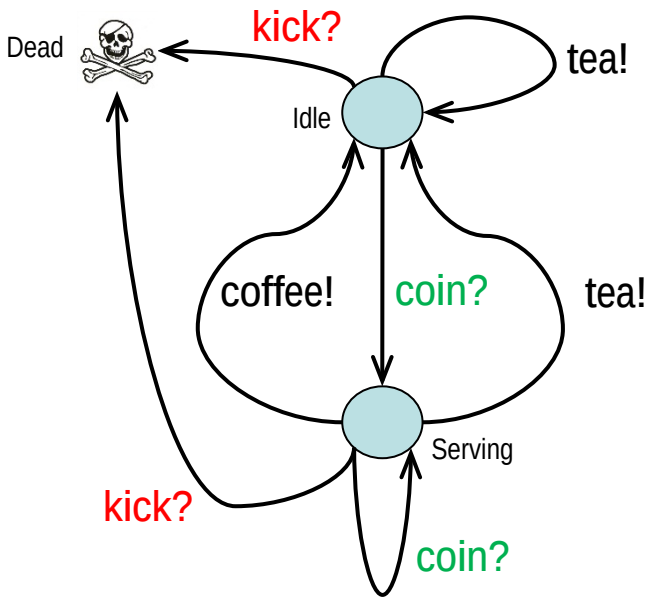


Concrete instances of the meta-theory

Component model

Input-output Automata

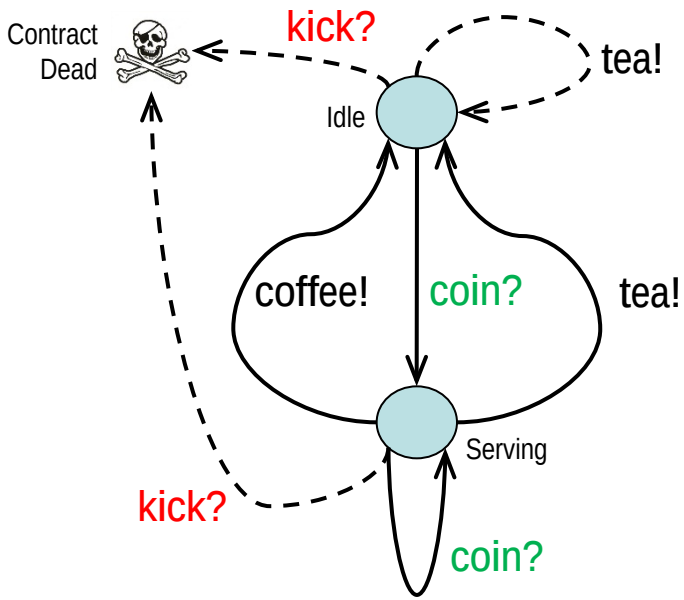
[Lynch]

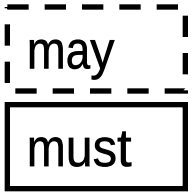


Contract model

Modal Interfaces

[Larsen & al] [Raclet & al]



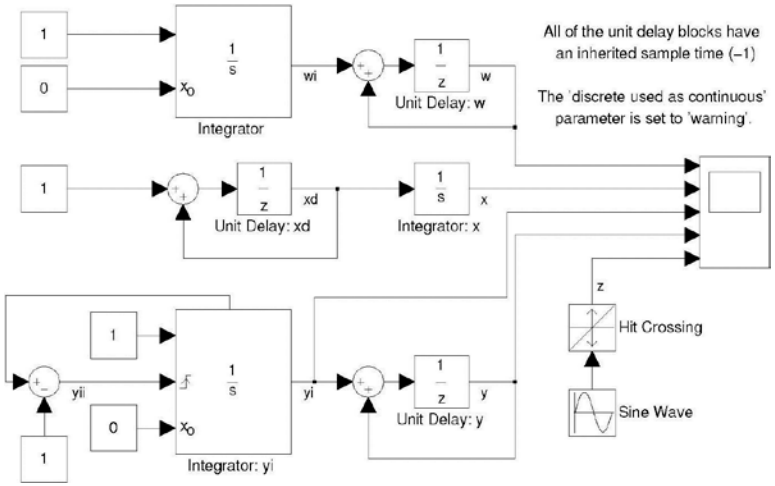


Concrete instances of the meta-theory

Component model

Dataflow diagram

[Simulink, Scade]

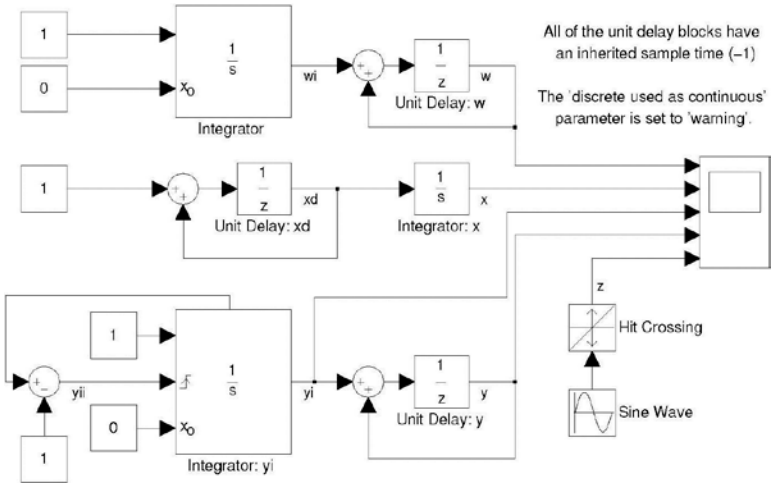


Concrete instances of the meta-theory

Component model

Dataflow diagram

[Simulink, Scade]



Contract model

Assume/Guarantee contracts

[Benveniste & al]

C = (A,G)

= (assumption,guarantee)

- A = assertion (diagram)
- G = assertion (diagram)

(E,M) ⊨ C

1. E satisfies A
2. E×M satisfies G

What is guaranteed is “A⇒G”

Concrete instances of the meta-theory

Component model	Contract model
Systems with non-trivial data	Observers, Abstract Interpretation
Timed systems	Restrictions needed
Probabilistic systems (reliability)	In progress

A toy illustrative example

A Parking Garage

Uses the MICA tool for Modal Interfaces by Benoît Caillaud

The Parking Garage toy example

- Top-level specification: requirements document
  - generic gate
  - payment machine
  - supervisor
- Different formalisms are used (textual and automata)
- Responsibilities assigned for each requirement $\begin{cases} \text{assumption} \\ \text{guarantee} \end{cases}$
- Requirements are translated into Modal Interfaces

The Parking Garage toy example

- Top-level specification: requirements document
  - generic gate
  - payment machine
  - supervisor
- Different formalisms are used (textual and automata)
- Responsibilities assigned for each requirement $\begin{cases} \text{assumption} \\ \text{guarantee} \end{cases}$
- Requirements are translated into Modal Interfaces

- Consistency, Compatibility, Correctness, Completeness
- Top-level Requirements ($\wedge$) $\longrightarrow$ Architecture of sub-systems ($\otimes$)
- Each sub-system can be submitted to a different supplier

The Parking Garage toy example

Top-level specification: assumptions & guarantees

gate(x) where $x \in \{\text{entry}, \text{exit}\}$

- $R_{g.1}(x)$: "vehicles shall not pass when x_gate is closed",
- $R_{g.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden
- $R_{g.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

payment

- $R_{p.1}$: "user inserts a coin every time a ticket is inserted and only then"
- $R_{p.2}$: "user may insert a ticket only initially or after an exit ticket has been issued"
- $R_{p.3}$: "exit ticket is issued after ticket is inserted and payment is made and only then"

supervisor

- $R_{g.1}$(entry)
- $R_{g.1}$(exit)
- $R_{g.2}$(entry)
- $R_{g.2}$(exit)
- $R_{s.1}$: initially and after !entry_gate close !entry_gate open is forbidden
- $R_{s.2}$: after !ticket_issued !entry_gate open must be enabled
- $R_{s.3}$: "at most one ticket is issued per vehicle entering the parking and tickets can be issued only if requested and ticket is issued only if the parking is not full"
- $R_{s.4}$: "when the entry gate is closed, the entry gate may not open unless a ticket has been issued"
- $R_{s.5}$: "the entry gate must open when a ticket is issued"
- $R_{s.6}$: "exit gate must open after an exit ticket is inserted and only then"
- $R_{s.7}$: "exit gate closes only after vehicle has exited parking"

The Parking Garage toy example

Top-level specification

gate(x) where $x \in$ {entry, exit}

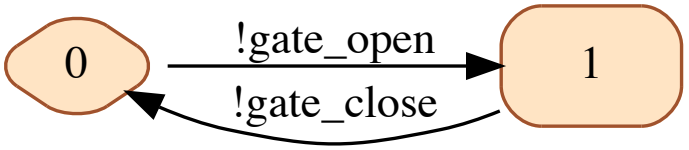
$R_{g.1}(x)$: "vehicles shall not pass when x_gate is closed",

$R_{g.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden

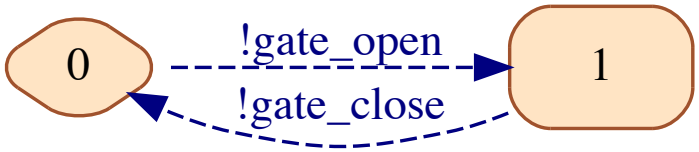
$R_{g.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the guarantees:

$R_{g.3}$ as an i/o-automaton:



$R_{g.3}$ as a Modal Interface:



The Parking Garage toy example

Top-level specification

gate(x) where $x \in$ {entry, exit}

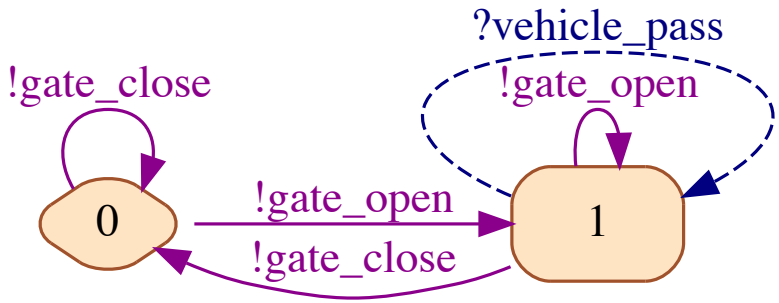
$R_{g.1}(x)$: "vehicles shall not pass when x_gate is closed",

$R_{g.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden

$R_{g.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the assumptions:

$R_{g.1}(x)$:



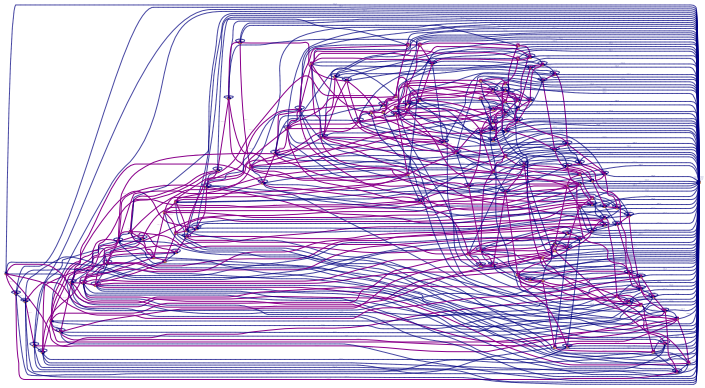
The Parking Garage toy example

Top-level specification: $\mathcal{C} = \mathcal{C}_{\textbf{gate}} \wedge \mathcal{C}_{\textbf{payment}} \wedge \mathcal{C}_{\textbf{supervisor}}$

gate
payment
supervisor

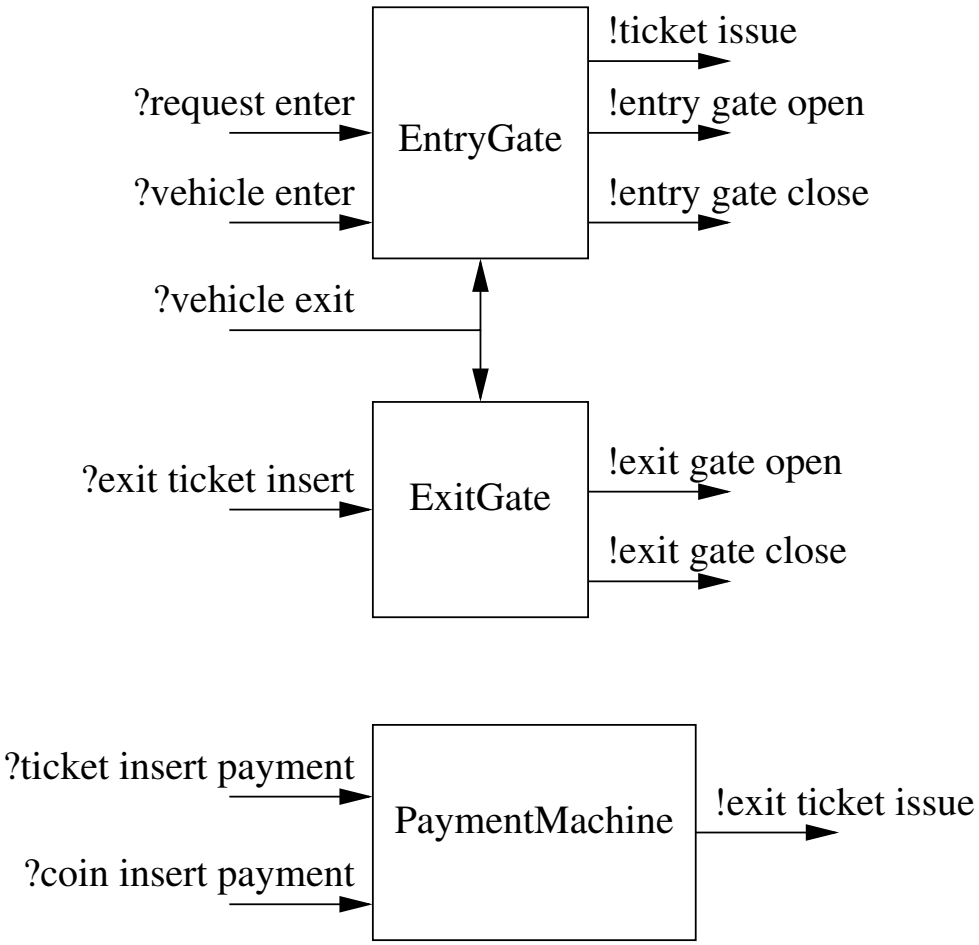
- $R_{g.1}$(entry)
- $R_{g.1}$(exit)
- $R_{g.2}$(entry)
- $R_{g.2}$(exit)
- $R_{s.1}$:initially and after !entry_gate close !entry_gate open is forbidden
- $R_{s.2}$:after !ticket_issued !entry_gate open must be enabled
- $R_{s.3}$:“at most one ticket is issued per vehicle entering the parking and tickets can be issued only if requested and ticket is issued only if the parking is not full”
- $R_{s.4}$:“when the entry gate is closed, the entry gate may not open unless a ticket has been issued”
- $R_{s.5}$:“the entry gate must open when a ticket is issued”
- $R_{s.6}$:“exit gate must open after an exit ticket is inserted and only then”
- $R_{s.7}$:“exit gate closes only after vehicle has exited parking”

$\mathcal{C}_{\textbf{supervisor}}$:



The Parking Garage toy example

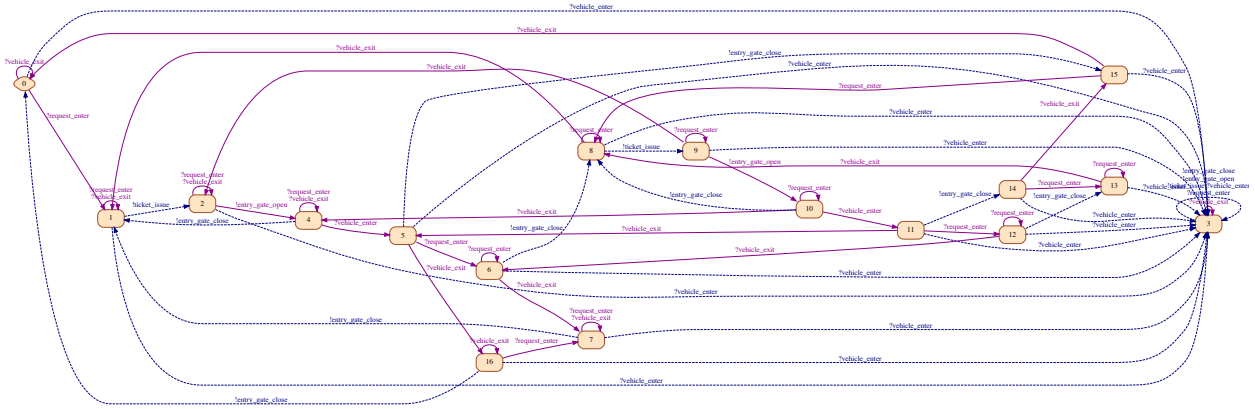
Architecture for sub-contracting $\mathcal{C}$ as a $\otimes$-composition of sub-systems (this is the duty of the designer)

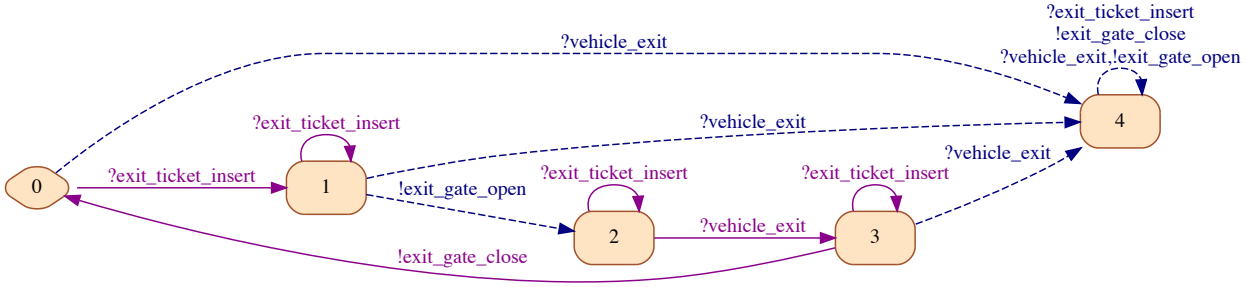


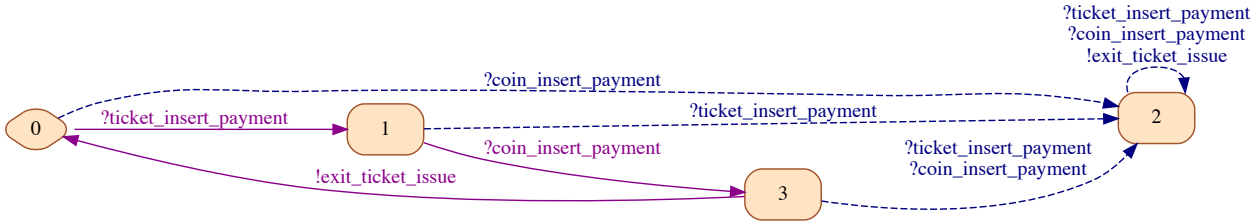
observe that the supervisor is distributed among the gates

The Parking Garage toy example

The following $\otimes$-decomposition of $\mathcal{C}$ into three sub-contracts was automatically generated (note the reduction in size)







Some needed research issues

- Observation, optimization & control problems as part of requirements?
- Requirements in natural language?

Observation, optimization & control problems as part of requirements?

Software for monitoring the physical system

- Requirements must specify
  - **Decision logic**
  - Sampling period
  - When to activate/inhibit monitoring
  - Time lag before fault confirmation
  - Fault identifier
  - Fault compensation
  - Returns to the cockpit
  - *Very diverse in nature*
- **Requirements must also specify *at a higher specification level* what the objective of the detection is, which fault to detect and isolate**
  decision logic ≈ implementation, low level

Observation, optimization & control problems as part of requirements?

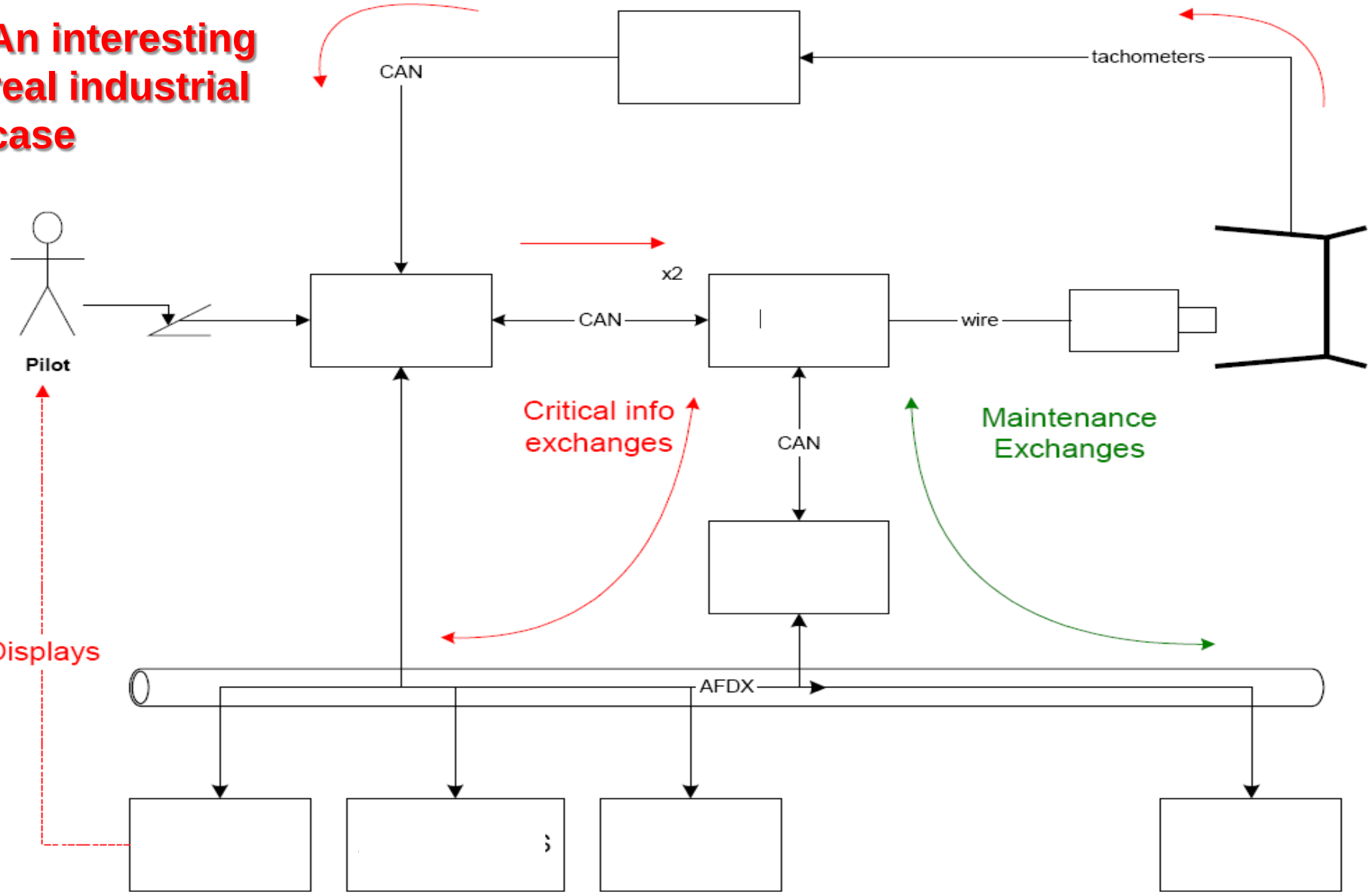
When the Engine Run Park mode is controlled **and** the park status information is FALSE and park_release_is_running is FALSE, the XXX shall :

- control each YYY to hold its position and energize all friction brakes, until the friction brakes are unlocked.
- **then**, apply a braking command with a braking force reference (F_BR) equal to C_ERP_CMD, until all valid YYYs have reached the braking command (braking_command_reached).
- **then**, command each YYY to hold its position and de-energize all friction brakes, until the friction brakes are locked.
- **then**, command each YYY to switch to the YYY Standby state.
- **AND** set B_POWERREQUEST_OUT to false**.**

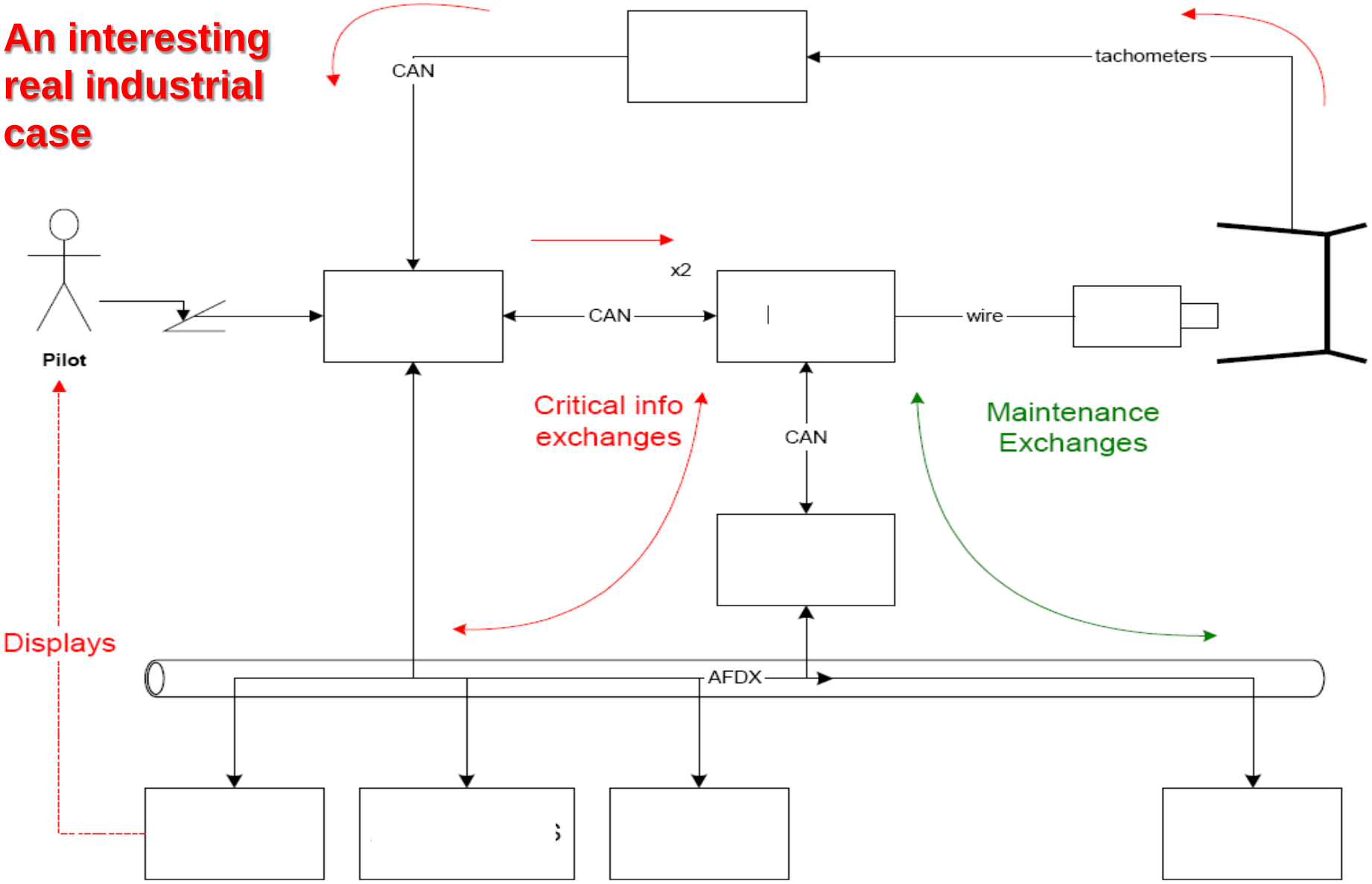
Seems quite low-level…
prefer control design problem formulation?

Observation, optimization & control problems as part of requirements?

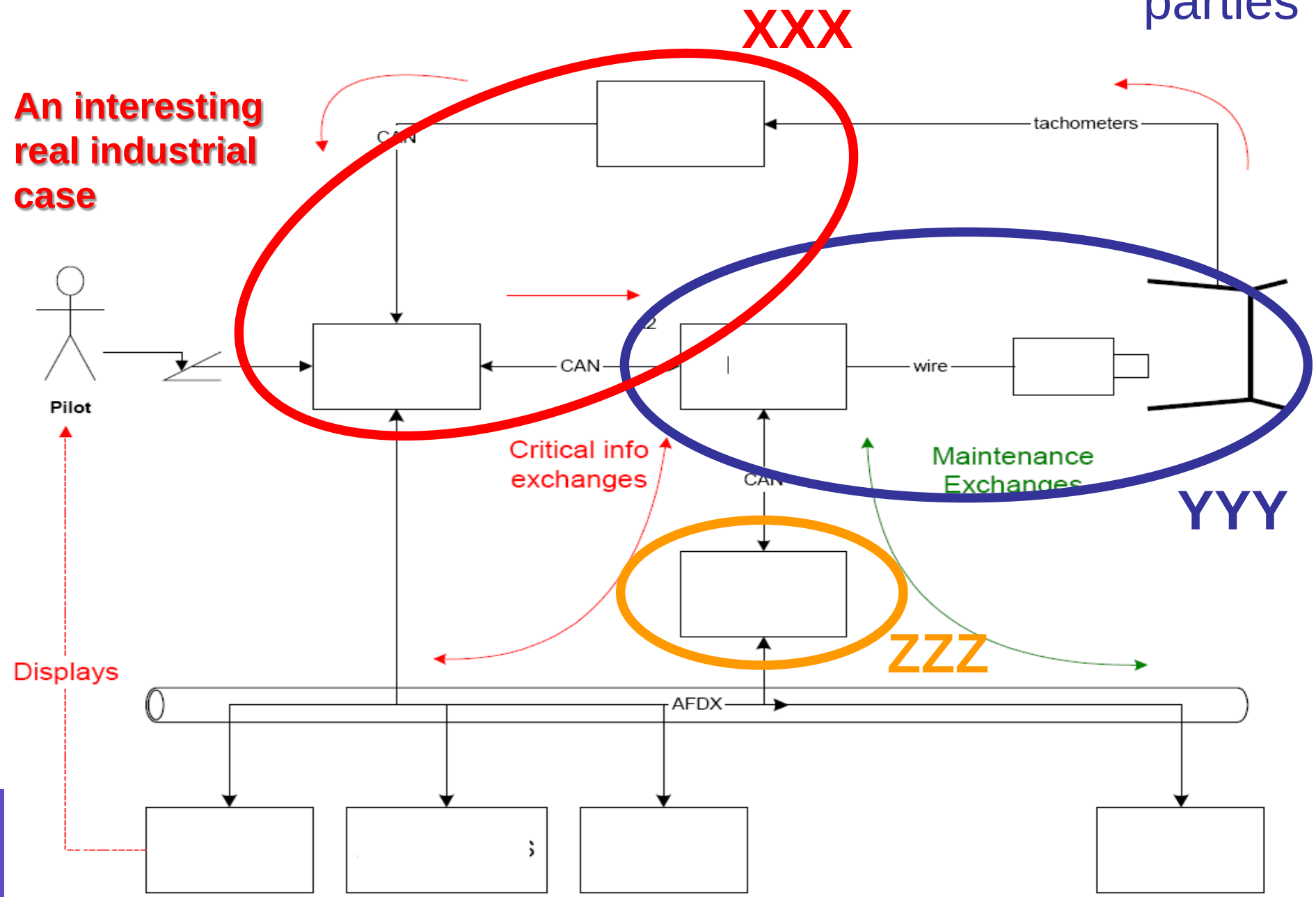
An interesting real industrial case



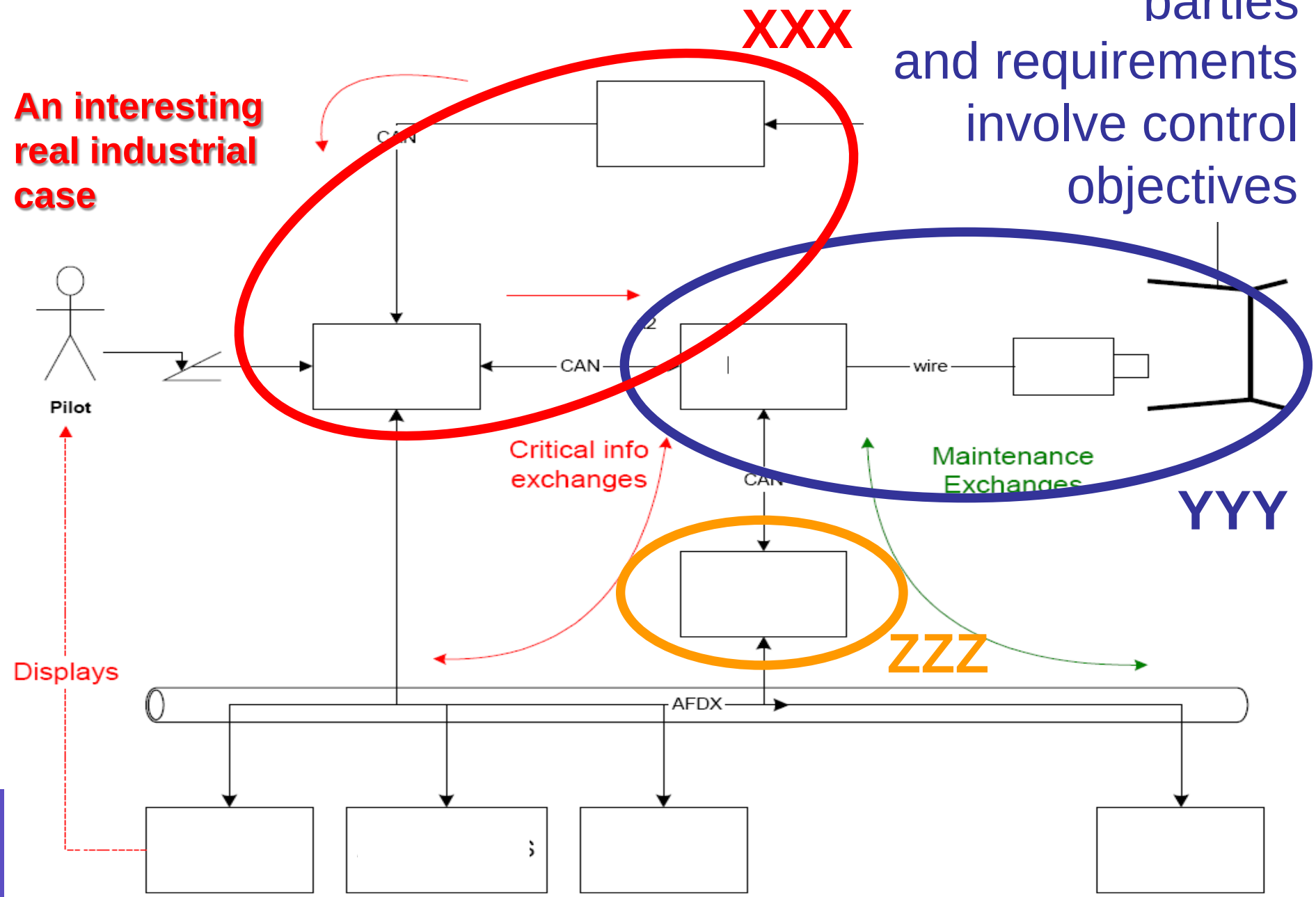
This real system involves a global feedback loop



It has, however, been subcontracted to 3 different parties



It has, however, been subcontracted to 3 different parties and requirements involve control objectives



Observation, optimization & control problems as part of requirements?

An interesting real industrial case

Splitting parts of the closed loop for development by different suppliers could be done, probably because requirements are already low-level, in the form of rules

Would not be acceptable if requirements were formulated at higher level of control design

Observation, optimization & control problems as part of requirements?

Today

Design of monitoring and control outside the scope of requirements management

Requirements engineering makes no use of mathematical techniques

Observation, optimization & control problems as part of requirements?

Today	Needed
Design of monitoring and control outside the scope of requirements management	Have monitoring and control within the scope of requirements management
Requirements engineering makes no use of mathematical techniques	Make it possible to have control or monitoring design problems embedded into requirements
	What is *implementation*, What is *refinement* ?

Some needed research issues

- Observation, optimization & control problems as part of requirements?
- Requirements in natural language?

Requirements in (semi)natural language?

Monitoring of	Test logic	Conf	Period	Test mode / act. phase	Sanction name	Treatment before conf	inhib. by	Environment data
RDC CAN Bus monitoring (Lost data during 3s)	if GNDTST_DATALOADSW is not refreshed	3s	1s	CBIT (permanent)	Use default value for GNDTST_DATALOADSW	Use the last valid GNDTST_DATALOADSW data	-	Brake_Intern_Context
Breaker	If during 40 AS frames [the status B_BREAKER_STAT4_IN is False (= Breaker4 opened) and B_WATCHDOG_DSP4_IN is False] 14 times or more	-	5ms	CBIT : (100ms after B_INV4_INHIB_OUT is set to TRUE) AND BOT_STATUS_IN different from 0x0F AND IT_STATUS_IN different from 0x0F	INHIB_EBA4	-	-	EBA4_Intern_Context

Requirements in (semi)natural language?

Today	Would be nice to have
Textual, limited control, Word, Excel, Doors	Handle all requirements in a unified way ⇒ natural language
Alternative: formal requirements (e.g., PSL in hardware) restricted use ⇒ different formalisms for different viewpoints	Requirements management based on natural language understanding.
	Seems unrealistic? MMhhh??

Conclusion

Requirements engineering is important

Interface and Contract theories have been very nice contributions from the research community

Achieving good matching

- Do not confine to decidable problems, accept testing; verification is a plus, possibly achieved by performing abstractions
- \+ Open theories to quantitative & other issues: synchronous, data, time, probas, QoS [Larsen, Legay, Caillaud, Raclet…]
- ? Lift up the level of specs by supporting, as part of requirements, mathematical formulations of *observation, control,* and *optimization* problems
- ? Requirements are diverse, theories are constrained: how to reconcile?