Contracts and Interfaces in the context of Requirements Engineering

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

1. Industrial context
2. Requirements on requirements engineering
3. Requirements by contracts: a meta-theory
4. Quick overview of Interface and Contract theories
5. An illustrative example
6. Conclusion
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, \( \approx 3000 \) requirements

Approximately 200 engineers in-house

Major sub-contractors \( \sim 10, 100 \) 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 :
- Calculatrices
- Réseaux de données
- Logiciels

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

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

- Requirements
- System architecture (PLM)
- Orchestrator
- Versioning
- Linking
- Design flow

Control design (Simulink)
Embedded electronics architecture (AADL)
Safety and reliability
Mapping data to wires and functions to ECU
Co-modeling & simu of functions and computing infrastructure
Testing V&V

System integration
Migrating from virtual to real
Validation

Orchestrator

Requirements Linking Design flow

System architecture (PLM)

Co-modeling & simu of functions and computing infrastructure
Mapping data to wires and functions to ECU
Testing V&V

Safety and reliability

(AUTOSAR platform)
A sample tool chain: links to requirements

Controls
design
(Simulink)

Embedded electronics architecture (AADL)

Safety and reliability

Mapping data to wires and functions to ECU

Co-modeling & simu of functions and computing infrastructure

Testing V&V

Orchestrator
Versioning
Linking
Design flow

Requirements

System architecture

(PLM)

System integration

Migrating from virtual to real

Validation

Links to testing and V&V

AUTOSAR platform
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 \(\sim\) implementation*
## Real examples of requirements from the monitoring viewpoint

<table>
<thead>
<tr>
<th>Monitoring of</th>
<th>Test logic</th>
<th>Conf</th>
<th>Period</th>
<th>Test mode / act. phase</th>
<th>Sanction name</th>
<th>Treatment before conf</th>
<th>inhib. by</th>
<th>Environment data</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDC CAN Bus</td>
<td>monitoring (Lost data during 3s) if GNDTST_DATALOADSW is not refreshed</td>
<td>3s</td>
<td>1s</td>
<td>CBIT (permanent)</td>
<td>Use default</td>
<td>Use the last valid</td>
<td></td>
<td>Brake_Intern_Context</td>
</tr>
<tr>
<td>Breaker</td>
<td>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</td>
<td>-</td>
<td>5ms</td>
<td>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</td>
<td>INHIB_EBA4</td>
<td>-</td>
<td></td>
<td>EBA4_Intern_Context</td>
</tr>
</tbody>
</table>
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

- System architecture
- Specification and development
- Requirements document
- Viewpoint
- Viewpoint
- Req1
- Req2
- ...

Developed by different teams

Structure:
- Context
- System

Semantics:
The Landscape: sub-contracting

- **Context**: System architecture
- **Structure**: System, Sub-system, Component
- **Semantics**: Specification and development, Requirements document, Viewpoint, Design

Developed by different suppliers and teams.
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

Structure:
- Context
- System architecture
- Sub-system
- Sub-system
- Component

Semantics:
- Specification and development
- Requirements document
- Viewpoint
- Requirements document
- Viewpoint
- Requirements document
- Design
- Design
- Design
- Designed System

Developed by different teams
Developed by different suppliers
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

structure

- System architecture
- Sub-system
- Sub-system
- Component

semantics

- Specification and development
- Requirements document
- Viewpoint
- Requirements document
- Viewpoint
- Requirements document
- Requirements document
- Design
- Design
- Design

Designed System

Developed by different suppliers
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

By different suppliers

\begin{align*}
\wedge \quad \text{Requirements document} \\
\quad \text{Viewpoint} \quad \text{Viewpoint} \\
\end{align*}

\begin{align*}
\quad \text{Requirements document} \\
\quad \text{Requirements document} \\
\quad \text{Requirements document} \\
\end{align*}

\begin{align*}
\text{Design} \\
\text{Design} \\
\text{Design} \\
\text{Designed System}
\end{align*}
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

By different suppliers

\[\text{Designed System}\]
Let’s go for the maths…
The meta-theory

- We assume some primitive concepts

<table>
<thead>
<tr>
<th>Component</th>
<th>$M$ open or closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composability</td>
<td>A type property</td>
</tr>
<tr>
<td>Composition</td>
<td>$\times$ commutative &amp; associative</td>
</tr>
<tr>
<td>Environment</td>
<td>$E : E \times M$ closed</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>Contract</th>
<th>$C = (\mathcal{E}_c, \mathcal{M}_c) = (\text{set of environments, set of implementations})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
<td>$\mathcal{M}_c \neq \emptyset$</td>
</tr>
<tr>
<td>Compatibility</td>
<td>$\mathcal{E}_c \neq \emptyset$</td>
</tr>
<tr>
<td>Implementation</td>
<td>$M \models^M C$ iff $M \in \mathcal{M}_c$ ; $E \models^E C$ iff $E \in \mathcal{E}_c$</td>
</tr>
</tbody>
</table>


## The meta-theory

### Generic Relations and Operators:

<table>
<thead>
<tr>
<th>Contract</th>
<th>( C = (\mathcal{E}_c, \mathcal{M}_c) = ) (set of environments , set of implementations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
<td>( \mathcal{M}_c \neq \emptyset )</td>
</tr>
<tr>
<td>Compatibility</td>
<td>( \mathcal{E}_c \neq \emptyset )</td>
</tr>
<tr>
<td>Implementation</td>
<td>( M \models^M C ) iff ( M \in \mathcal{M}_c ); ( E \models^E C ) iff ( E \in \mathcal{E}_c )</td>
</tr>
<tr>
<td>Refinement</td>
<td>( C' \preceq C ) iff ( \mathcal{E}_{c'} \supseteq \mathcal{E}<em>c ) and ( \mathcal{M}</em>{c'} \subseteq \mathcal{M}_c )</td>
</tr>
<tr>
<td>Conjunction</td>
<td>( C_1 \land C_2 = \text{GLB for } \preceq )</td>
</tr>
</tbody>
</table>
## The meta-theory

- **Generic Relations and Operators:**

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract</td>
<td>$C = (E_C, M_C) = (\text{set of environments}, \text{set of implementations})$</td>
</tr>
<tr>
<td>Consistency</td>
<td>$M_C \neq \emptyset$</td>
</tr>
<tr>
<td>Compatibility</td>
<td>$E_C \neq \emptyset$</td>
</tr>
<tr>
<td>Implementation</td>
<td>$M \models^M C \iff M \in M_C$ ; $E \models^E C \iff E \in E_C$</td>
</tr>
<tr>
<td>Refinement</td>
<td>$C' \preceq C \iff E_{C'} \supseteq E_C$ and $M_{C'} \subseteq M_C$</td>
</tr>
<tr>
<td>Conjunction</td>
<td>$C_1 \land C_2 = \text{GLB for } \preceq$</td>
</tr>
<tr>
<td>Composition</td>
<td>$C_1 \otimes C_2 = \bigwedge \left{ C \mid \begin{align*} [M_1 \models^M C_1 \text{ and } M_2 \models^M C_2] &amp; \Rightarrow M_1 \times M_2 \models^M C \ E \models^E C &amp; \Rightarrow [E \times M_2 \models^E C_1 \text{ and } E \times M_1 \models^E C_2] \end{align*} \right}$</td>
</tr>
</tbody>
</table>
## The meta-theory

- **Generic Relations and Operators:**

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract</td>
<td>$C = (\mathcal{E}_c, \mathcal{M}_c) = (\text{set of environments, set of implementations})$</td>
</tr>
<tr>
<td>Consistency</td>
<td>$\mathcal{M}_c \neq \emptyset$</td>
</tr>
<tr>
<td>Compatibility</td>
<td>$\mathcal{E}_c \neq \emptyset$</td>
</tr>
<tr>
<td>Implementation</td>
<td>$M \models^M C$ iff $M \in \mathcal{M}_c$ ; $E \models^E C$ iff $E \in \mathcal{E}_c$</td>
</tr>
<tr>
<td>Refinement</td>
<td>$C' \preceq C$ iff $\mathcal{E}_{c'} \supseteq \mathcal{E}<em>c$ and $\mathcal{M}</em>{c'} \subseteq \mathcal{M}_c$</td>
</tr>
<tr>
<td>Conjunction</td>
<td>$C_1 \land C_2 = \text{GLB for } \preceq$</td>
</tr>
<tr>
<td>Composition</td>
<td>$C_1 \otimes C_2 = \bigwedge \left{ C \mid \right. \left. \begin{array}{l} [M_1 \models^M C_1 \text{ and } M_2 \models^M C_2] \Rightarrow M_1 \times M_2 \models^M C \ \text{and} \ E \models^E C \Rightarrow [E \times M_2 \models^E C_1 \text{ and } E \times M_1 \models^E C_2] \end{array} \right}$</td>
</tr>
</tbody>
</table>
| Quotient       | $C_1 / C_2 = \max\{ C \mid C \otimes C_2 \preceq C_1 \}$ }
# The meta-theory

- **Generic Properties:**

<table>
<thead>
<tr>
<th>Refinement</th>
<th>substituability of environments substituability of implementations</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Composition</th>
<th>$(C_1, C_2)$ compatible $C'_i \leq C_i \quad \Rightarrow \quad {\ (C'_1, C'_2) \text{ compatible } \ C'_1 \otimes C'_2 \leq C_1 \otimes C_2 }$ independent implementability</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$(C_1 \otimes C_2) \otimes (C_3 \otimes C_4) = (C_1 \otimes C_3) \otimes (C_2 \otimes C_4)$ compatibility $\Rightarrow$ compatibility associativity</th>
</tr>
</thead>
</table>

| Quotient | $C \leq C_1/C_2 \iff C \otimes C_2 \leq C_1$ |

- **Distributivity** (freedom in design processes)
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

<table>
<thead>
<tr>
<th>Component model</th>
<th>Contract model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems with non-trivial data</td>
<td>Observers, Abstract Interpretation</td>
</tr>
<tr>
<td>Timed systems</td>
<td>Restrictions needed</td>
</tr>
<tr>
<td>Probabilistic systems (reliability)</td>
<td>In progress</td>
</tr>
</tbody>
</table>
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 \(\left\{\begin{array}{l}
\text{assumption} \\
\text{guarantee}
\end{array}\right.\)

- 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 \(\{\text{assumption}\, \text{guarantee}\}\)
- Requirements are translated into Modal Interfaces

- Consistency, Compatibility, Correctness, Completeness
- Top-level Requirements \(\land\) \(\rightarrow\) 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 \(x \in \{\text{entry, exit}\}\)
- \(R_{g.1}(x)\): “vehicles shall not pass when \(x\_\text{gate is closed}\),
- \(R_{g.2}(x)\): 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**
- \(R_{g.1}(\text{entry})\)
- \(R_{g.1}(\text{exit})\)
- \(R_{g.2}(\text{entry})\)
- \(R_{g.2}(\text{exit})\)
- \(R_{s.1}\): 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 ∈ {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**

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

- \( R_{g.1}(x) \): “vehicles shall not pass when \( x \_ \text{gate is closed} \),
- \( R_{g.2}(x) \): 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:**

\[
\begin{align*}
R_{g.1}(x): & \quad 0 \xrightarrow{!\text{gate_close}} 1 \\
R_{g.2}(x): & \quad 0 \xrightarrow{?\text{vehicle_pass}} 1 \\
R_{g.3}: & \quad 0 \xrightarrow{!\text{gate_open}} 1
\end{align*}
\]
The Parking Garage toy example

Top-level specification: \( C = C_{\text{gate}} \land C_{\text{payment}} \land C_{\text{supervisor}} \)

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 \( ! \text{entry\_gate\_close} \) \( ! \text{entry\_gate\_open} \) is forbidden
\( R_{s.2} \) : after \( ! \text{ticket\_issued} \) \( ! \text{entry\_gate\_open} \) 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"

\( C_{\text{supervisor}} \) :
The Parking Garage toy example

Architecture for sub-contracting $C$ 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 $C$ 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 BPOWERREQUEST_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

An interesting real industrial case
It has, however, been subcontracted to 3 different parties

An interesting real industrial case

Displays

Critical info exchanges

Maintenance Exchanges

Pilot

YYY

XXX

ZZZ

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

An interesting real industrial case
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?

<table>
<thead>
<tr>
<th>Today</th>
<th>Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design of monitoring and control outside the scope of requirements management</td>
<td>Have monitoring and control within the scope of requirements management</td>
</tr>
<tr>
<td>Requirements engineering makes no use of mathematical techniques</td>
<td>Make it possible to have control or monitoring design problems embedded into requirements</td>
</tr>
<tr>
<td></td>
<td>What is <em>implementation</em>, What is <em>refinement</em>?</td>
</tr>
</tbody>
</table>
Some needed research issues

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

<table>
<thead>
<tr>
<th>Monitoring of</th>
<th>Test logic</th>
<th>Conf</th>
<th>Period</th>
<th>Test mode / act. phase</th>
<th>Sanction name</th>
<th>Treatment before conf</th>
<th>inhib. by</th>
<th>Environment data</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDC CAN Bus monitoring (Lost data during 3s)</td>
<td>if GNDTST_DATALOADSW is not refreshed</td>
<td>3s</td>
<td>1s</td>
<td>CBIT (permanent)</td>
<td>Use default value for GNDTST_DATALOADSW</td>
<td>Use the last valid GNDTST_DATALOADSW data</td>
<td>-</td>
<td>Brake_Intern_Context</td>
</tr>
<tr>
<td>Breaker</td>
<td>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</td>
<td>-</td>
<td>5ms</td>
<td>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</td>
<td>INHIB_EBA4</td>
<td>-</td>
<td>-</td>
<td>EBA4_Intern_Context</td>
</tr>
</tbody>
</table>
## Requirements in (semi)natural language?

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