# Contract Theories

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

INRIA Rennes

April 17, 2015

# Contracts in embedded systems design: why?

- Formalizing OEM/supplier relations: “contracts for contracts”

  Complement Legal Contracts with Technical Contracts

- Structuring requirements or specifications

  Requirements are structured into chapters/viewpoints/aspects
  (function, safety, performance & timing, QoS. . . )

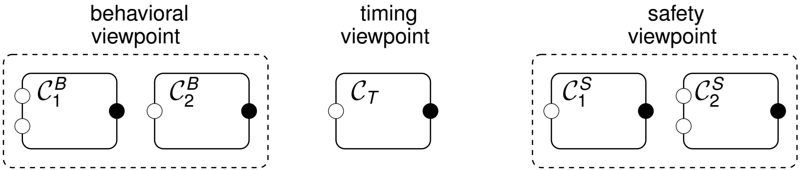
- Concurrent development at the system designer

  Different viewpoints are developed by different teams
  Weaving viewpoints must be sound and correct

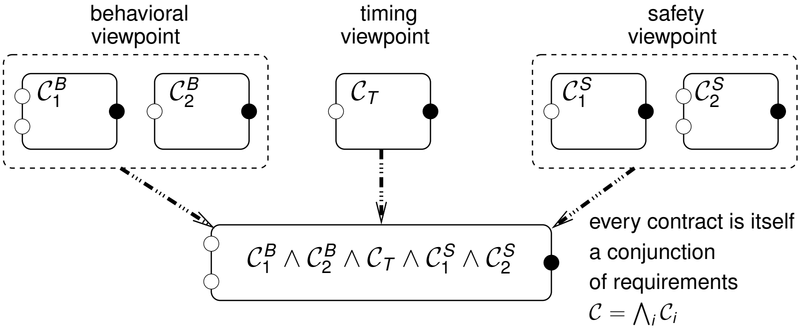
- Independent development by the suppliers

  Suppliers must be able to develop their sub-systems having
  all the info they need; system integration must be correct

# Structuring requirements or specifications
# Concurrent development



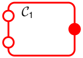
# Structuring requirements or specifications
# Concurrent development



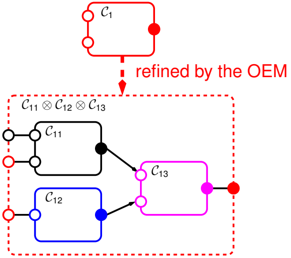

Requirements are combined by using “contract conjunction”

Viewpoints are fused by using “contract conjunction”

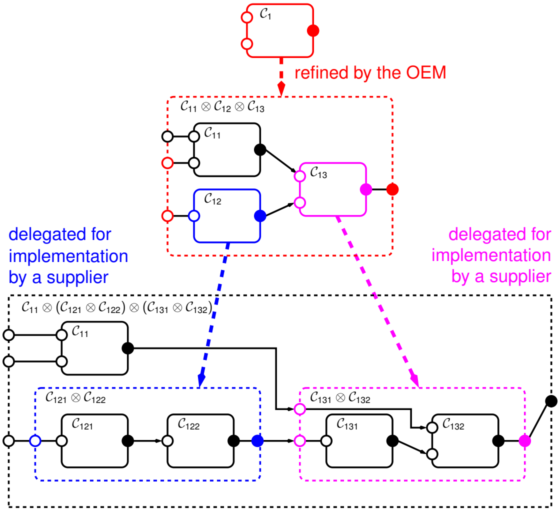
# Structuring requirements or specifications
# Independent development



# Structuring requirements or specifications
# Independent development



# Structuring requirements or specifications
# Independent development




“refined”, “implementation”, $\otimes$: new concepts

**A meta-theory of contracts**

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Motivations for a meta-theory

A wide and diverse bibliography:

- Systems from components:
  OO-programming in the 80's [B. Meyer. . . ]
- Refinement
  - by simulation: in the 80's [Milner], OK for closed systems
  - by alternating simulation for open systems:
    early 90's [Abadi, Lamport, Wolper]
    late 90's [de Alfaro, Kupferman, Henzinger]
- Composition and compatibility:
  [Abadi, Lamport93] [de Alfaro-Henzinger 2000]
- Conjunction and consistency:
  [Passerone, Raclet, Caillaud, Benveniste... 2008]
- Product lines (not discussed here): [Larsen, Nyman, Wasowski 2008]

# Motivations for a meta-theory

Fact:

- Different frameworks have been proposed to address similar issues:
  - specification theories
  - interface theories
  - contract theories

**the meta-theory**

Goal:

- Capture the essence of the above frameworks
- Highlight their very nature
- Develop new generic tools and techniques
- Instantiate to known frameworks, hoping for new results

# The meta-theory: Components and Contracts

- Components: actual pieces of SW/HW/devices, open system
- Environment: context of use (a component), often unkown at design time
- Components cannot constrain their environment

# The meta-theory: Components and Contracts

- Components: actual pieces of SW/HW/devices, open system
- Environment: context of use (a component), often unkown at design time
- Components cannot constrain their environment

- Contracts are intentionally abstract
- Pinpoint responsibilities of component vs. environment

$$\boxed{\text{semantics}(\mathcal{C}) = \left( \underbrace{\mathcal{E}_{\mathcal{C}}}_{\substack{\text{set of}\\ \text{environments}}} , \underbrace{\mathcal{M}_{\mathcal{C}}}_{\substack{\text{set of}\\ \text{components}}} \right)}$$

# The meta-theory

- We assume some primitive concepts:

| | | |
|---|---|---|
| Component | $M$ | |
| Composition | $\times$ | is partially defined, commutative and associative |
| Composability | $M_1 \times M_2$ | being well-defined is a typing relation |
| Environment | $E$ | is an environment for $M$ iff $E \times M$ is well-defined |

- On top of these primitive concepts we define
  - generic concepts and operators
  - satisfying generic properties

- How concepts, operators, and properties, are made effective
  - depends on the specific framework

# The meta-theory

- Generic Relations and Operators:

| | |
|---|---|
| Contract | sem $(\mathcal{C}) = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ where $\mathcal{C} \in \mathbf{C}$ : underlying class of contracts |
| Consistency | $\mathcal{M}_{\mathcal{C}} \neq \emptyset$ |
| Compatibility | $\mathcal{E}_{\mathcal{C}} \neq \emptyset$ |
| | |
| | |
| | |
| | |
| | |

# The meta-theory

- Generic Relations and Operators:

| | |
|---|---|
| Contract | $\mathrm{sem}(\mathcal{C}) = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ where $\mathcal{C} \in \mathbf{C}$ : underlying class of contracts |
| Consistency | $\mathcal{M}_{\mathcal{C}} \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is consistent iff $\mathcal{C}_1 \wedge \mathcal{C}_2$ is consistent |
| Compatibility | $\mathcal{E}_{\mathcal{C}} \neq \emptyset$ |
| Implementation | $M \models^{\mathsf{M}} \mathcal{C}$ iff $M \in \mathcal{M}_{\mathcal{C}}$ ; $E \models^{\mathsf{E}} \mathcal{C}$ iff $E \in \mathcal{E}_{\mathcal{C}}$ |
| Refinement | $\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{\mathcal{C}'} \supseteq \mathcal{E}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{C}'} \subseteq \mathcal{M}_{\mathcal{C}}$ |
| Conjunction | $\mathcal{C}_1 \wedge \mathcal{C}_2 =$ GLB for $\preceq$ within $\mathbf{C}$<br>$\mathcal{C}_1 \vee \mathcal{C}_2 =$ LUB for $\preceq$ within $\mathbf{C}$ |
| | |
| | |

# The meta-theory

- Generic Relations and Operators:

| | |
|---|---|
| Contract | $\text{sem}(\mathcal{C}) = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ where $\mathcal{C} \in \mathbf{C}$ : underlying class of contracts |
| Consistency | $\mathcal{M}_{\mathcal{C}} \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is consistent iff $\mathcal{C}_1 \wedge \mathcal{C}_2$ is consistent |
| Compatibility | $\mathcal{E}_{\mathcal{C}} \neq \emptyset$ say that $(\mathcal{C}_1, \mathcal{C}_2)$ is compatible iff $\mathcal{C}_1 \otimes \mathcal{C}_2$ is compatible |
| Implementation | $M \models^{\mathrm{M}} \mathcal{C}$ iff $M \in \mathcal{M}_{\mathcal{C}}$ ; $E \models^{\mathrm{E}} \mathcal{C}$ iff $E \in \mathcal{E}_{\mathcal{C}}$ |
| Refinement | $\mathcal{C}' \preceq \mathcal{C}$ iff $\mathcal{E}_{\mathcal{C}'} \supseteq \mathcal{E}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{C}'} \subseteq \mathcal{M}_{\mathcal{C}}$ |
| Conjunction | $\mathcal{C}_1 \wedge \mathcal{C}_2$ = GLB for $\preceq$ within $\mathbf{C}$<br>$\mathcal{C}_1 \vee \mathcal{C}_2$ = LUB for $\preceq$ within $\mathbf{C}$ |
| Composition | $\mathcal{C}_1 \otimes \mathcal{C}_2 = \min \left\{ \mathcal{C} \;\middle|\; \begin{bmatrix} \forall M_1 & \models^{\mathrm{M}} & \mathcal{C}_1 \\ \forall M_2 & \models^{\mathrm{M}} & \mathcal{C}_2 \\ \forall E & \models^{\mathrm{E}} & \mathcal{C} \end{bmatrix} \Rightarrow \begin{bmatrix} M_1 \times M_2 & \models^{\mathrm{M}} & \mathcal{C} \\ E \times M_2 & \models^{\mathrm{E}} & \mathcal{C}_1 \\ E \times M_1 & \models^{\mathrm{E}} & \mathcal{C}_2 \end{bmatrix} \right\}$ |
| Quotient | $\mathcal{C}_1 / \mathcal{C}_2 = \max\{ \mathcal{C} \in \mathbf{C} \mid \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1 \}$ |

# The meta-theory

Generic Properties:

| | |
|---|---|
| Refinement | substituability $\nearrow$ of sets of environments<br>substituability $\searrow$ of sets of implementations |
| Composition | $\left.\begin{array}{r}(\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \\ \mathcal{C}'_i \preceq \mathcal{C}_i\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(\mathcal{C}'_1, \mathcal{C}'_2) \text{ compatible} \\ \mathcal{C}'_1 \otimes \mathcal{C}'_2 \preceq \mathcal{C}_1 \otimes \mathcal{C}_2\end{array}\right.$<br>independent implementability<br><br>$(\mathcal{C}_1 \otimes \mathcal{C}_2) \otimes \mathcal{C}_3 = \mathcal{C}_1 \otimes (\mathcal{C}_2 \otimes \mathcal{C}_3)$<br>associativity<br><br>$[(\mathcal{C}_{11} \wedge \mathcal{C}_{21}) \otimes (\mathcal{C}_{12} \wedge \mathcal{C}_{22})] \preceq [(\mathcal{C}_{11} \otimes \mathcal{C}_{12}) \wedge (\mathcal{C}_{21} \otimes \mathcal{C}_{22})]$<br>sub-distributivity: sets the freedom in design processes, fusing viewpoints before/after composing sub-systems |
| Quotient | $\mathcal{C} \preceq \mathcal{C}_1 / \mathcal{C}_2 \;\Leftrightarrow\; \mathcal{C} \otimes \mathcal{C}_2 \preceq \mathcal{C}_1$ |

# Abstracting and testing

- Restrictions must hold for relations and operators on contracts to be analyzable
- Such restrictions may not hold for system models in practice
- Typical obstacles are infinite data types and functions operating on them

# Abstracting and testing

- Restrictions must hold for relations and operators on contracts to be analyzable
- Such restrictions may not hold for system models in practice
- Typical obstacles are infinite data types and functions operating on them

- Two complementary ways of overcoming this consist in
  - performing abstractions
  - performing testing

- The meta-theory offers generic means:
  - abstraction on top of abstract interpretation for components
  - observers for contract-compliant testing

# Bibliographical note

Very few attempts to develop a meta-theory. Two recent papers:

- Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifiations to contracts in component-based design. FASE 2012
  - Starts from an abstract notion of specification with axioms—refinement, conjunction, composition, quotient
  - Then it defines contracts as pairs $(A, G)$ of specs
  - It establishes a link from abstract specs to modal automata
- Taolue Chen, Chris Chilton, Bengt Jonsson, Marta Z. Kwiatkowska: A Compositional Specification Theory for Component Behaviours. ESOP 2012: 148-168
  - trace based abstract specification
  - ports split into uncontrolled/controlled (or input/output)
  - assumptions involve inputs and guarantees involve outputs
  - conjunction, composition, quotient

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Abstracting and Testing contracts $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$

Approach:

1. Assume a Galois connection on components
2. Yields a canonical abstraction on sets of components
3. Yields a canonical abstraction for contracts

Properties:

- Consistency and Compatibility can be proved on abstractions (positive semi-decision)
- Contract abstraction is monotonic with respect to refinement
- Contract abstraction distributes over conjunction
- Contract abstraction "sub-distributes" over composition

There are obstructions to getting an abstraction with stronger properties

# Abstracting and Testing contracts $\mathcal{C} = (\mathcal{E}_c, \mathcal{M}_c)$

1. Following [Cousot&Cousot], recall the notion of Galois connection:

$$\begin{array}{lll} \alpha : (\mathcal{X}_{\mathbf{C}}, \sqsubseteq_{\mathbf{C}}) \mapsto (\mathcal{X}_{\mathbf{A}}, \sqsubseteq_{\mathbf{A}}) & : & \text{the abstraction} \\ \gamma : (\mathcal{X}_{\mathbf{A}}, \sqsubseteq_{\mathbf{A}}) \mapsto (\mathcal{X}_{\mathbf{C}}, \sqsubseteq_{\mathbf{C}}) & : & \text{the concretization} \end{array}$$

   two monotonic maps such that

$$X_{\mathbf{C}} \sqsubseteq_{\mathbf{C}} \gamma(X_{\mathbf{A}}) \iff \alpha(X_{\mathbf{C}}) \sqsubseteq_{\mathbf{A}} X_{\mathbf{A}}$$

2. From Galois connection on $X$'s to abstractions on sets-of-$X$:
   - Let $\mathcal{X}^{<} \subseteq 2^{\mathcal{X}}$ collect all $\sqsubseteq$-downward closed subsets of $\mathcal{X}$
   - Equip $\mathcal{X}_{\mathbf{C}}^{<}$ and $\mathcal{X}_{\mathbf{A}}^{<}$ with their inclusion orders $\subseteq_{\mathbf{C}}$ and $\subseteq_{\mathbf{A}}$
   - Set

$$\widehat{\alpha}(\chi_{\mathbf{C}}) = \gamma^{-1}(\chi_{\mathbf{C}}) = \{X_{\mathbf{A}} \mid \gamma(X_{\mathbf{A}}) \text{ well defined and } \in \chi_{\mathbf{C}}\}$$

3. the canonical way of defining abstractions for contracts is:

$$\alpha(\mathcal{C}_{\mathbf{C}}) = \left(\widehat{\alpha}(\mathcal{E}_{c_{\mathbf{C}}}), \widehat{\alpha}(\mathcal{M}_{c_{\mathbf{C}}})\right)$$

# Abstracting and Testing contracts $\mathcal{C} = (\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$

Approach:

1. Assume a testing technique on components
2. An observer for contracts is thus a pair of tests (for environments and components, respectively)

Properties:

- Implementations can be disproved using observers (negative semi-decision)
- Observers for contract conjunction can be obtained compositionally
- Observers for contract composition can “almost” be obtained compositionally

# Abstracting and Testing contracts $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$

An observer for $\mathcal{C}$ is a pair $(b^{\mathrm{E}}_{\mathcal{C}}, b^{\mathrm{M}}_{\mathcal{C}})$ of non-deterministic boolean functions $\mathcal{M} \mapsto \{\mathit{false}, \mathit{true}\}$ called verdicts, such that:

$$\left.\begin{array}{lcl} b^{\mathrm{E}}_{\mathcal{C}}(E) \text{ outputs } \mathit{false} & \Longrightarrow & E \notin \mathcal{E}_\mathcal{C} \\ b^{\mathrm{M}}_{\mathcal{C}}(M) \text{ outputs } \mathit{false} & \Longrightarrow & M \notin \mathcal{M}_\mathcal{C} \end{array}\right\} \text{semi-decision}$$

| Operator | Observer |
|---|---|
| $\mathcal{C} = (\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$ | $(b^{\mathrm{E}}_{\mathcal{C}}, b^{\mathrm{M}}_{\mathcal{C}})$ |
| $\mathcal{C} = \mathcal{C}_1 \wedge \mathcal{C}_2$ | $b^{\mathrm{E}}_{\mathcal{C}} = b^{\mathrm{E}}_{\mathcal{C}_1} \vee b^{\mathrm{E}}_{\mathcal{C}_2}$ , $b^{\mathrm{M}}_{\mathcal{C}} = b^{\mathrm{M}}_{\mathcal{C}_1} \wedge b^{\mathrm{M}}_{\mathcal{C}_2}$ |
| $\mathcal{C} = \mathcal{C}_1 \vee \mathcal{C}_2$ | $b^{\mathrm{E}}_{\mathcal{C}} = b^{\mathrm{E}}_{\mathcal{C}_1} \wedge b^{\mathrm{E}}_{\mathcal{C}_2}$ , $b^{\mathrm{M}}_{\mathcal{C}} = b^{\mathrm{M}}_{\mathcal{C}_1} \vee b^{\mathrm{M}}_{\mathcal{C}_2}$ |
| $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ | $b^{\mathrm{E}}_{\mathcal{C}}(E) = \bigwedge_{\substack{M_1 \models^{\mathrm{M}} \mathcal{C}_1 \\ M_2 \models^{\mathrm{M}} \mathcal{C}_2}} \left[ b^{\mathrm{E}}_{\mathcal{C}_2}(E \times M_1) \wedge b^{\mathrm{E}}_{\mathcal{C}_1}(E \times M_2) \right]$ <br> $b^{\mathrm{M}}_{\mathcal{C}}(M_1 \times M_2) = b^{\mathrm{M}}_{\mathcal{C}_1}(M_1) \wedge b^{\mathrm{M}}_{\mathcal{C}_2}(M_2)$ |

A meta-theory of contracts
Details
Meta-theory $\mapsto$ Assume/Guarantee contracts
Details
Meta-theory $\mapsto$ Interface Automata
Details
Meta-theory $\mapsto$ Modal Interfaces
Details
Contract Based Requirement Engineering
Details
Concluding Remarks

# Assume/Guarantee contracts: summary

- Component:
  - Kahn Process Network (KPN) or
  - Synchronous Transition System (STS)
- Contract: pair $(\mathrm{Assumption}, \mathrm{Guarantee}) =$ (KPN,KPN) or (STS,STS)

  $\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:

$$
\begin{array}{rcl}
\mathcal{E}_{\mathcal{C}} & = & \{E \mid E \subseteq A\} \\
\mathcal{M}_{\mathcal{C}} & = & \{M \mid A \times M \subseteq G\}
\end{array}
$$

# Assume/Guarantee contracts: summary

- Component:
  - Kahn Process Network (KPN) or
  - Synchronous Transition System (STS)
- Contract: pair $(\text{Assumption}, \text{Guarantee}) =$ (KPN,KPN) or (STS,STS)
  $\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:

$$\begin{array}{rcl} \mathcal{E}_{\mathcal{C}} & = & \{E \mid E \subseteq A\} \\ \mathcal{M}_{\mathcal{C}} & = & \{M \mid A \times M \subseteq G\} \end{array}$$

- The following (existing) definitions specialize the meta-theory:

$$\begin{array}{rcl} \mathcal{C}_1 \wedge \mathcal{C}_2 & \equiv & (A_1 \cup A_2, G_1 \cap G_2) \\ \mathcal{C}_1 \otimes \mathcal{C}_2 & \equiv & ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2) \end{array}$$

# Assume/Guarantee contracts: summary

- Component:
  - Kahn Process Network (KPN) or
  - Synchronous Transition System (STS)
- Contract: pair $(\text{Assumption}, \text{Guarantee}) = (\text{KPN},\text{KPN})$ or $(\text{STS},\text{STS})$
  $\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$ following the meta-theory:

$$\begin{array}{rcl} \mathcal{E}_\mathcal{C} & = & \{E \mid E \subseteq A\} \\ \mathcal{M}_\mathcal{C} & = & \{M \mid A \times M \subseteq G\} \end{array}$$

- The following (existing) definitions specialize the meta-theory:

$$\begin{array}{rcl} \mathcal{C}_1 \wedge \mathcal{C}_2 & \equiv & (A_1 \cup A_2, G_1 \cap G_2) \\ \mathcal{C}_1 \otimes \mathcal{C}_2 & \equiv & ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2) \end{array}$$

- No quotient exists
- Dealing with variable alphabets of variables is unsatisfactory, due to an unfortunate handling of assumptions in contract conjunction

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# A/G contracts: the Component Model

- To simplify we present the theory for a fixed alphabet $\Sigma$ and without distinguishing input vs. output
- A/G contract theory builts on top of component models that are assertions (sets of behaviors). We can consider both
  - asynchronous Kahn Process Networks (KPN)
  - Synchronous Transition Systems (synchronous languages)
  - for both cases, parallel composition is by intersection

$$
\begin{array}{rcl}
M & = & (\Sigma, P) \;=\; P \quad \text{for short since } \Sigma \text{ is fixed} \\
P & \subseteq & \left\{ \begin{array}{lll} & \Sigma \mapsto \ \text{Dom}^* \cup \text{Dom}^\omega & \text{KPN} \\ \text{or} & & \\ & (\Sigma \mapsto \text{Dom})^* \cup (\Sigma \mapsto \text{Dom})^\omega & \text{Synchronous} \end{array} \right. \\
M_1 \times M_2 & = & P_1 \cap P_2
\end{array}
$$

# A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; $A$ (the assumptions) and $G$ (the guarantees) are assertions over $\Sigma$

Behaviors must be of the same kind for both components and contracts
(either both KPN or both synchronous)

# A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; $A$ (the assumptions) and $G$ (the guarantees) are assertions over $\Sigma$

Behaviors must be of the same kind for both components and contracts (either both KPN or both synchronous)

$\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_\mathcal{C}, \mathcal{M}_\mathcal{C})$, where:

$$\begin{aligned}
\mathcal{E}_\mathcal{C} &= \{E \mid E \subseteq A\} \\
\mathcal{M}_\mathcal{C} &= \{M \mid A \times M \subseteq G\}
\end{aligned}$$

# A/G contracts: the Contracts

$\mathcal{C} = (A, G)$; $A$ (the assumptions) and $G$ (the guarantees) are assertions over $\Sigma$

Behaviors must be of the same kind for both components and contracts (either both KPN or both synchronous)

$\mathcal{C} = (A, G)$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$, where:

$$\begin{array}{rcl} \mathcal{E}_{\mathcal{C}} & = & \{E \mid E \subseteq A\} \\ \mathcal{M}_{\mathcal{C}} & = & \{M \mid A \times M \subseteq G\} \end{array}$$

Contracts $\mathcal{C}$ and $\mathcal{C}'$ such that

$$A = A' \quad \text{and} \quad G \cup \neg A \; = \; G' \cup \neg A'$$

are equivalent as they yield identical sets of environments and components.

$\mathcal{C}$ can always be saturated meaning that $G \supseteq \neg A$. This is assumed next.

# A/G contracts: the Contracts

Refinement (for $\mathcal{C}_1, \mathcal{C}_2$ saturated):

$$\mathcal{C}' \preceq \mathcal{C} \text{ holds iff } \left\{ \begin{array}{l} A' \supseteq A \\ G' \subseteq G \end{array} \right.$$

Composition (for $\mathcal{C}_1, \mathcal{C}_2$ saturated):

$$\begin{array}{rcl} G & = & G_1 \cap G_2 \\ A & = & \max \left\{ A \;\middle|\; \begin{array}{l} A \cap G_2 \subseteq A_1 \\ A \cap G_1 \subseteq A_2 \end{array} \right\} \quad = \quad (A_1 \cap A_2) \cup \neg(G_1 \cap G_2) \end{array}$$

No quotient exists

**Problem**: need to complement assertions. Use observers or abstractions?

Abstractions and observers can be defined

# A/G contracts with variable alphabet

Variable alphabet is dealt with using a two steps procedure

1. Equalize alphabets in both assumptions and guarantees
   (by existential inverse projection)
2. Reuse the theory developed for a fixed alphabet

Whereas alphabet equalization is known and works well for components (and environments), we do have a problem when extending it to contracts:

**Problem**: alphabet equalization for A/G-contracts is well defined but is practically inadequate when dealing with the conjunction, as it yields, for the assumptions and when alphabets are disjoint:

$$A_1 \cup A_2 = \mathit{true}$$

meaning that every environment is considered legal

# Bibliographical note

- A/G reasoning arises in OO-programming in the late 80's [B. Meyer 1992]
  Contracts quite often deal with complex typing handled with constraints expressed on parameters (OCL)

- Formal behavioral contracts come in the early 90's in the area of compositional verification; main issue here is that of *circular reasoning* [Clarke, de Long, Mc Millan 1989]; see also [Abadi & Lamport 1993]

- A/G behavioral contracts were revisited in [Passerone et al. 2007] by SPEEDS project

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

**Meta-theory $\mapsto$ Interface Automata**

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Interface Automata: summary

- Component: deterministic and receptive input/output automaton
  - $M = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q \ni q_0, \rightarrow)$ with usual parallel composition $M_1 \times M_2$
- Contract: deterministic (possibly non receptive) input/output automaton
  - $\mathcal{C} = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \rightarrow)$
  - $\mathcal{C}$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:
    - $\mathcal{E}_{\mathcal{C}}$ collects all $E$ not proposing as output an action that is refused by $\mathcal{C}$ in the composition $E \times \mathcal{C}$
    - $\mathcal{M}_{\mathcal{C}}$ collects all $M$ such that, $\forall E \in \mathcal{E}_{\mathcal{C}}$, $E \times \mathcal{C}$ simulates $E \times M$

# Interface Automata: summary

- Refinement, as defined by alternating simulation, specializes the meta-theory;

  Conjunction is difficult, even for a fixed alphabet of actions

- Parallel composition $\otimes$, together with its notion of compatibility, exist and specialize the meta-theory:

  1. start from $\mathcal{C}_1 \times \mathcal{C}_2$, seen as i/o-automata
  2. illegal pair $(q_1, q_2)$ may exist, where (informally) "$\mathcal{M}_{\mathcal{C}_1} \not\subseteq \mathcal{E}_{\mathcal{C}_2}$"
  3. pruning illegal pairs until fixpoint yields $\mathcal{C}_1 \otimes \mathcal{C}_2$

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

**Details**

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Interface Automata

We only present the theory for a fixed alphabet $\Sigma$
Alphabet equalization is as for A/G contracts (with the same problems)

Interface theories built on top of (a relaxed version of) Nancy Lynch i/o-automata

Deterministic Input/Output Automaton: $M = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \rightarrow)$, where:

$$\begin{array}{rcl}
\Sigma = \Sigma^{\mathrm{in}} \cup \Sigma^{\mathrm{out}} & : & \text{alphabet of actions} \\
q_0 \in Q & : & \text{initial state} \\
q \xrightarrow{\alpha} q' & : & \text{deterministic transition relation} \left.\begin{array}{l} q \xrightarrow{\alpha} q_1 \\ q \xrightarrow{\alpha} q_2 \end{array}\right\} \Rightarrow q_1 = q_2
\end{array}$$

# Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\mathrm{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

# Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\mathrm{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

- Parallel composition: well defined only if $\Sigma_1^{\mathrm{out}} \cap \Sigma_2^{\mathrm{out}} = \emptyset$; the two components synchronize on their actions

$$M_1 \times M_2 : \left\{ \begin{array}{rcl} \Sigma^{\mathrm{out}} & = & \Sigma_1^{\mathrm{out}} \cup \Sigma_2^{\mathrm{out}} \\ Q & = & Q_1 \times Q_2 \\ q_0 & = & (q_{1,0}, q_{2,0}) \\ (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2') & \text{iff} & q_i \xrightarrow{\alpha}_i q_i', i = 1, 2 \end{array} \right.$$

# Interface Automata: the Component Model

- Component: an i/o-automaton that is receptive:

$$\forall q \in Q, \forall \alpha \in \Sigma^{\mathrm{in}}, \exists q' : \; q \xrightarrow{\alpha} q'$$

- Parallel composition: well defined only if $\Sigma_1^{\mathrm{out}} \cap \Sigma_2^{\mathrm{out}} = \emptyset$; the two components synchronize on their actions

$$M_1 \times M_2 : \left\{ \begin{array}{rcl} \Sigma^{\mathrm{out}} & = & \Sigma_1^{\mathrm{out}} \cup \Sigma_2^{\mathrm{out}} \\ Q & = & Q_1 \times Q_2 \\ q_0 & = & (q_{1,0}, q_{2,0}) \\ (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2') & \text{iff} & q_i \xrightarrow{\alpha}_i q_i', i = 1, 2 \end{array} \right.$$

- Simulation: For $q_i \in Q_i$, say that $q_2 \leq q_1$ if

$$\forall \alpha, q_2' \text{ such that } q_2 \xrightarrow{\alpha}_2 q_2' \quad \Longrightarrow \quad \left\{ \begin{array}{l} q_1 \xrightarrow{\alpha}_1 q_1' \\ \text{and } q_2' \leq q_1' \end{array} \right.$$

Say that $M_2 \leq M_2$ if $q_{2,0} \leq q_{1,0}$

# Interface Automata: Contracts

Interface Automaton $\mathcal{C} = (\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, q_0, \rightarrow)$

- $\Sigma^{\text{in}}, \Sigma^{\text{out}}, Q, \rightarrow$ as in i/o-automata
- We do not request $q_0 \in Q$; thus, $q_0 \notin Q$ is also a possibility

When $q_0 \in Q$, $\mathcal{C}$ defines a contract by fixing a pair $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$, where:

- $\mathcal{E}_{\mathcal{C}}$ collects all $E$ such that:
  1. $\Sigma^{\text{in}}_E = \Sigma^{\text{out}}$ and $\Sigma^{\text{out}}_E = \Sigma^{\text{in}}$. Thus, $E$ and $\mathcal{C}$, seen as i/o-automata, are composable
  2. $E$ is $\mathcal{C}$-compliant:
  $$\left.\begin{array}{r} \forall \alpha \in \Sigma^{\text{out}}_E, q_E \xrightarrow{\alpha}_E \\ \forall (q_E, q) \text{ reachable in } E \times \mathcal{C} \end{array}\right\} \Rightarrow (q_E, q) \xrightarrow{\alpha}_{E \times \mathcal{C}} \text{ holds}$$
- $\mathcal{M}_{\mathcal{C}}$ collects all $M$ such that $\forall E \in \mathcal{E}_{\mathcal{C}}, \mathcal{C}$ simulates $E \times M$ seen as i/o-automata

**Lemma:** $q_0 \in Q$ iff $\mathcal{C}$ is both consistent ($\mathcal{M}_{\mathcal{C}} \neq \emptyset$) and compatible ($\mathcal{E}_{\mathcal{C}} \neq \emptyset$)

# Interface Automata: Contracts

Refinement is by alternating simulation: for $\mathcal{C}_1, \mathcal{C}_2$ two contracts such that $\Sigma_1^{\mathrm{out}} = \Sigma_2^{\mathrm{out}}$, say that $q_2 \preceq q_1$ if

$$\forall \alpha \in \Sigma_2^{\mathrm{out}}, \forall q_2' \text{ s.t. } q_2 \xrightarrow{\alpha}_2 q_2' \implies \begin{cases} q_1 \xrightarrow{\alpha}_1 q_1' \\ q_2' \preceq q_1' \end{cases}$$

$$\forall \alpha \in \Sigma_1^{\mathrm{in}}, \forall q_1' \text{ s.t. } q_1 \xrightarrow{\alpha}_1 q_1' \implies \begin{cases} q_2 \xrightarrow{\alpha}_2 q_2' \\ q_2' \preceq q_1' \end{cases}$$

Say that $\mathcal{C}_2 \preceq \mathcal{C}_1$ if $q_{2,0} \preceq q_{1,0}$.

Conjunction exists but is difficult.

# Interface Automata: Contracts

Parallel Composition for $\mathcal{C}_1, \mathcal{C}_2$ two contracts such that $\Sigma_1^{\mathrm{out}} \cap \Sigma_2^{\mathrm{out}} = \emptyset$:

1. Build $\mathcal{C}_1 \times \mathcal{C}_2$ as an i/o-automaton and try it as the composition

2. By the meta-theory we must have

$$E \models^{\mathsf{E}} \mathcal{C} \quad \Longrightarrow \quad \left[ E \times M_2 \models^{\mathsf{E}} \mathcal{C}_1 \text{ and } E \times M_1 \models^{\mathsf{E}} \mathcal{C}_2 \right]$$

   which requires $\forall \alpha \in \Sigma_i^{\mathrm{out}}\colon\ q_i \xrightarrow{\alpha}_i \quad \Longrightarrow \quad (q_1, q_2) \xrightarrow{\alpha}_{\mathcal{C}_2 \times \mathcal{C}_1}$

   $(q_1, q_2)$ not satisfying this is called illegal and must be pruned away

3. Perform this recursively until fixpoint $\mathcal{C}_1 \otimes \mathcal{C}_2$; the resulting $Q$ can be empty:

$$\begin{array}{lcl} Q \text{ empty} & \Longleftrightarrow & (\mathcal{C}_1, \mathcal{C}_2) \text{ incompatible} \\ Q \text{ nonempty} & \Longleftrightarrow & (\mathcal{C}_1, \mathcal{C}_2) \text{ compatible} \end{array}$$

# Bibliographical note

[de Alfaro Henzinger], several papers in the early 2000's

- Composition and compatibility
- Refinement by alternating simulation
- State based models and Variable based models
- Conjunction (called shared refinement by the authors) is more delicate…

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

**Meta-theory $\mapsto$ Modal Interfaces**

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Modal Interfaces: Summary

- Component: deterministic and receptive input/output automaton
  - $M = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q{\ni}q_0, \rightarrow)$ with usual parallel composition $M_1 \times M_2$
- Contract: $\mathcal{C} = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \underbrace{\rightarrow}_{\text{must}}, \underbrace{\dashrightarrow}_{\text{may}})$
  - $\mathcal{C}$ yields $(\mathcal{C}^{must}, \mathcal{C}^{may})$, deterministic non-receptive i/o-automata
  - $\mathcal{C}$ defines a contract $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ following the meta-theory:
    - $\mathcal{E}_{\mathcal{C}}$ collects all $E$ not proposing as output an action that is refused by $\mathcal{C}^{must}$ in the composition $E \times \mathcal{C}^{must}$
    - $\mathcal{M}_{\mathcal{C}}$ collects all $M$ such that, $\forall E{\in}\mathcal{E}_{\mathcal{C}}$, $E \times \mathcal{C}^{may}$ simulates $E \times M$ and $E \times M$ simulates $E \times \mathcal{C}^{must}$
  - consistency condition: $\rightarrow \subseteq \dashrightarrow$

# Modal Interfaces: Summary

- Modal Refinement $\mathcal{C}_2 \preceq \mathcal{C}_1$, defined by

$$\begin{array}{rl} & \dashrightarrow_2 \subseteq \dashrightarrow_1 \\ \text{and} & \longrightarrow_1 \subseteq \longrightarrow_2 \end{array}$$

  specializes the meta-theory;

  Conjunction follows by pruning illegal pairs of states for consistency

- Parallel composition $\otimes$, together with its notion of compatibility, exist; Quotient exists; all specialize the meta-theory

- Variable alphabets are handled by using different alphabet equalizations
  - for $\wedge$: adding *may* self-loops, and
  - for $\otimes$: adding *may*$+$*must* self-loops

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Modal Interfaces: Components

- We first develop the theory for a fixed alphabet, and then the general case
- The model of components is the same as for Interface Automata, namely deterministic and receptive i/o-automata

# Modal Interfaces: Contracts

Modal Interface: $\mathcal{C} = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \rightarrow, \dashrightarrow)$:

- $\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0$ are as in Interface Automata ($q_0 \in Q$ may not hold)
- $\rightarrow, \dashrightarrow \subseteq Q \times \Sigma \times Q$ are two transition relations, called must and may

# Modal Interfaces: Contracts

Modal Interface: $\mathcal{C} = (\Sigma^{\mathrm{in}}, \Sigma^{\mathrm{out}}, Q, q_0, \rightarrow, \dashrightarrow)$:

When $q_0 \in Q$, $\mathcal{C}$ yields two (generally non receptive) i/o-automata denoted by $\mathcal{C}^{must}$ and $\mathcal{C}^{may}$ and defines $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$ as follows:

- $\mathcal{E}_{\mathcal{C}}$ collects all $E$ such that:
    1. $\Sigma^{\mathrm{in}}_E = \Sigma^{\mathrm{out}}$ and $\Sigma^{\mathrm{out}}_E = \Sigma^{\mathrm{in}}$; hence $E \times \mathcal{C}^{must}$ is well defined;
    2. $E$ is $\mathcal{C}^{must}$-compliant:

$$\left.\begin{array}{r} \forall \alpha \in \Sigma^{\mathrm{out}}_E \\ \forall q_E : q_E \xrightarrow{\alpha}_E \\ \forall q : (q_E, q) \text{ reachable in } E \times \mathcal{C}^{may} \end{array}\right\} \Rightarrow (q_E, q) \xrightarrow{\alpha}_{E \times \mathcal{C}^{must}}$$

- $\mathcal{M}_{\mathcal{C}}$ collects all $M$ such that, for any $E \in \mathcal{E}_{\mathcal{C}}$:
    3. only *may* transitions are allowed for $E \times M$: $\mathcal{C}^{may}$ simulates $E \times M$
    4. *must* transitions are mandatory in $E \times M$:

$$\left.\begin{array}{r} \forall (q_E, q_M) \text{ reachable in } E \times M \\ \forall q \in \mathcal{C}^{may} : (q_E, q_M) \leq q \\ \forall \alpha \in \Sigma^{\mathrm{out}}_M : q \xrightarrow{\alpha}_{\mathcal{C}^{must}} \end{array}\right\} \Rightarrow (q_E, q_M) \xrightarrow{\alpha}_{E \times M}$$

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \stackrel{\alpha}{\rightarrow} \ \Rightarrow q \stackrel{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists \alpha, q \stackrel{\alpha}{\rightarrow}$ but $q \stackrel{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \ : \ \text{prune } q \text{ away}$$

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists\alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \;:\; \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists \alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \; : \; \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$
2. Performing this makes state $q$ unreachable in $\mathcal{C}^{may}$; as a result, the set of environments is possibly augmented

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists\alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \; : \; \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$
2. Performing this makes state $q$ unreachable in $\mathcal{C}^{may}$; as a result, the set of environments is possibly augmented
3. Since we have removed may transitions, some more states have possibly become inconsistent. So, we repeat until fixpoint

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists\alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \ : \ \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$
2. Performing this makes state $q$ unreachable in $\mathcal{C}^{may}$; as a result, the set of environments is possibly augmented
3. Since we have removed may transitions, some more states have possibly become inconsistent. So, we repeat until fixpoint
4. At fixpoint, $Q = Q_{\mathsf{con}} \uplus Q_{\mathsf{incon}}$, collecting consistent and inconsistent states

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$ ; if $q$ inconsistent $\exists\alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$ , i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathsf{E}} \mathcal{C} \\ \forall M \models^{\mathsf{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \ : \ \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$
2. Performing this makes state $q$ unreachable in $\mathcal{C}^{may}$; as a result, the set of environments is possibly augmented
3. Since we have removed may transitions, some more states have possibly become inconsistent. So, we repeat until fixpoint
4. At fixpoint, $Q = Q_{\text{con}} \uplus Q_{\text{incon}}$, collecting consistent and inconsistent states
5. Since $Q_{\text{incon}}$ is unreachable from $Q_{\text{con}}$, delete $Q_{\text{incon}}$, thus obtaining $[\mathcal{C}]$

# Modal Interfaces: Consistency and Compatibility

Call state $q$ consistent if $q \xrightarrow{\alpha} \Rightarrow q \overset{\alpha}{\dashrightarrow}$; if $q$ inconsistent $\exists\alpha, q \xrightarrow{\alpha}$ but $q \overset{\alpha}{\not\dashrightarrow}$, i.e.:

$$\left.\begin{array}{l} \forall E \models^{\mathrm{E}} \mathcal{C} \\ \forall M \models^{\mathrm{M}} \mathcal{C} \end{array}\right\} \Rightarrow \text{ no } (q_E, q_M) \text{ of } E \times M \text{ satisfies } (q_E, q_M) \leq q \; : \; \text{prune } q \text{ away}$$

1. May transitions leading to $q$ can be erased without changing $(\mathcal{E}_{\mathcal{C}}, \mathcal{M}_{\mathcal{C}})$
2. Performing this makes state $q$ unreachable in $\mathcal{C}^{may}$; as a result, the set of environments is possibly augmented
3. Since we have removed may transitions, some more states have possibly become inconsistent. So, we repeat until fixpoint
4. At fixpoint, $Q = Q_{\text{con}} \uplus Q_{\text{incon}}$, collecting consistent and inconsistent states
5. Since $Q_{\text{incon}}$ is unreachable from $Q_{\text{con}}$, delete $Q_{\text{incon}}$, thus obtaining $[\mathcal{C}]$

**Theorem:**

1. $\mathcal{M}_{[\mathcal{C}]} = \mathcal{M}_{\mathcal{C}}$ and $\mathcal{E}_{[\mathcal{C}]} \supseteq \mathcal{E}_{\mathcal{C}}$
2. $[\mathcal{C}]$ offers the smallest set of environments with this property
3. $\mathcal{C}$ is consistent and compatible if and only if $Q_{\text{con}} \ni q_0$

# Modal Interfaces: Refinement and Conjunction

For $\mathcal{C}$ a Modal Interface and $q \in Q$ a state of it:

$$\begin{array}{rcl} may(q) & = & \{\alpha \in \Sigma \mid q \overset{\alpha}{\dashrightarrow} \} \\ must(q) & = & \{\alpha \in \Sigma \mid q \overset{\alpha}{\rightarrow} \ \} \end{array}$$

For $\mathcal{C}_i, i = 1, 2$ and $q_i$ a state of $\mathcal{C}_i$. Say that $q_2 \preceq q_1$, if:

$$\left\{ \begin{array}{rcl} may_2(q_2) & \subseteq & may_1(q_1) \\ must_2(q_2) & \supseteq & must_1(q_1) \end{array} \right.$$

$$\text{and } \forall \alpha \in \Sigma : \quad \left\{ \begin{array}{l} q_1 \overset{\alpha}{\dashrightarrow}_1 q_1' \\ q_2 \overset{\alpha}{\dashrightarrow}_2 q_2' \end{array} \right. \quad \Longrightarrow \quad q_2' \preceq q_1'$$

Say that $\mathcal{C}_2 \preceq \mathcal{C}_1$ if $q_{2,0} \preceq q_{1,0}$.

**Theorem:**

$$\mathcal{C}_2 \preceq \mathcal{C}_1 \quad \text{iff} \quad \left\{ \begin{array}{rcl} \mathcal{E}_{\mathcal{C}_2} & \supseteq & \mathcal{E}_{\mathcal{C}_1} \\ \mathcal{M}_{\mathcal{C}_2} & \subseteq & \mathcal{M}_{\mathcal{C}_1} \end{array} \right.$$

# Modal Interfaces: Refinement and Conjunction

For $\mathcal{C}$ a Modal Interface and $q \in Q$ a state of it:

$$\begin{array}{rcl} may(q) & = & \{\alpha \in \Sigma \mid q \overset{\alpha}{\dashrightarrow}\} \\ must(q) & = & \{\alpha \in \Sigma \mid q \overset{\alpha}{\rightarrow}\ \} \end{array}$$

The pre-conjunction $\mathcal{C}_1 \& \mathcal{C}_2$ of two Modal Interfaces is only defined if $\Sigma_1^{\mathrm{in}} = \Sigma_2^{\mathrm{in}}$ and $\Sigma_1^{\mathrm{out}} = \Sigma_2^{\mathrm{out}}$ and is given by:

$$\begin{array}{rcl} \Sigma^{\mathrm{in}} = \Sigma_1^{\mathrm{in}} & ; & \Sigma^{\mathrm{out}} = \Sigma_1^{\mathrm{out}} \\ Q = Q_1 \times Q_2 & ; & q_0 = (q_{1,0}, q_{2,0}) \\ must(q_1, q_2) & = & must_1(q_1) \cup must_2(q_2) \\ may(q_1, q_2) & = & may_1(q_1) \cap may_2(q_2) \end{array}$$

The conjunction is defined as

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \quad = \quad [\mathcal{C}_1 \& \mathcal{C}_2]$$

and is the GLB for refinement order: specializes the meta-theory.

# Modal Interfaces: Composition and Quotient

The composition $\mathcal{C}_1 \otimes \mathcal{C}_2$ of two Modal Interfaces is only defined if $\Sigma_1^{\text{out}} \cap \Sigma_2^{\text{out}} = \emptyset$

It is given by: $\Sigma^{\text{out}} = \Sigma_1^{\text{out}} \cup \Sigma_2^{\text{out}}$, $Q = Q_1 \times Q_2$, $q_0 = (q_{1,0}, q_{2,0})$, and:

$$\begin{array}{rcl} \mathit{must}[0](q_1, q_2) & = & \mathit{must}_1(q_1) \cap \mathit{must}_2(q_2) \\ \mathit{may}[0](q_1, q_2) & = & \mathit{may}_1(q_1) \cap \mathit{may}_2(q_2) \end{array}$$

Say that pair $(q_1, q_2)$ is illegal if

$$\begin{array}{rrcl} & \mathit{may}(q_1) \cap \Sigma_2^{\text{in}} & \not\subseteq & \mathit{must}(q_2) \\ \text{or} & \mathit{may}(q_2) \cap \Sigma_1^{\text{in}} & \not\subseteq & \mathit{must}(q_1) \end{array}$$

Illegal pairs of states cause harm to environments and must be pruned away. Pruning illegal pairs of states until fixpoint (as above) yields

$$\mathcal{C}_1 \otimes \mathcal{C}_2$$

specializing the meta-theory

# Modal Interfaces with variable alphabets

We consider Modal Interfaces with consistent states only

$$\& \quad : \quad \left\{ \begin{array}{rcl} must(q_1, q_2) & = & must_1(q_1) \cup must_2(q_2) \\ may(q_1, q_2) & = & may_1(q_1) \cap may_2(q_2) \end{array} \right.$$

$$\otimes \quad : \quad \left\{ \begin{array}{rcl} must(q_1, q_2) & = & must_1(q_1) \cap must_2(q_2) \\ may(q_1, q_2) & = & may_1(q_1) \cap may_2(q_2) \end{array} \right.$$

Alphabet equalization should be neutral against all relations and operations

$$\text{for } \& \quad : \quad \left[ \begin{array}{c} \alpha \in may_1(q_1) \text{ and } \alpha \in whatever_2(q_2) \\ \Downarrow \\ \alpha \in whatever(q_1, q_2) \end{array} \right]$$

$$\text{for } \otimes \quad : \quad \left[ \begin{array}{c} \alpha \in must_1(q_1) \text{ and } \alpha \in whatever_2(q_2) \\ \Downarrow \\ \alpha \in whatever(q_1, q_2) \end{array} \right]$$

Neutral alphabet extension is by adding

- *may* self-loops for the conjunction and
- *must*+*may* self-loops for the composition

# Modal Interfaces with variable alphabets

We consider Modal Interfaces with consistent states only
Define the weak extension of $\mathcal{C}$ to $\Sigma' \supset \Sigma$, written $\mathcal{C}^{\Uparrow\Sigma'}$, as follows:

$$\mathcal{C}^{\Uparrow\Sigma'} : \left\{ \begin{array}{rcl} (\Sigma^{\mathrm{in}})^{\Uparrow\Sigma'} & = & \Sigma^{\mathrm{in}} \cup (\Sigma' \setminus \Sigma) \\ (\Sigma^{\mathrm{out}})^{\Uparrow\Sigma'} & = & \Sigma^{\mathrm{out}} \\ Q^{\Uparrow\Sigma'} & = & Q \\ q_0^{\Uparrow\Sigma'} & = & q_0 \\ \mathit{may}^{\Uparrow\Sigma'} & = & \mathit{may} \cup (\Sigma' \setminus \Sigma) \\ \mathit{must}^{\Uparrow\Sigma'} & = & \mathit{must} \end{array} \right.$$

and the strong extension of $\mathcal{C}$ to $\Sigma' \supset \Sigma$, written $\mathcal{C}^{\uparrow\Sigma'}$

$$\mathcal{C}^{\uparrow\Sigma'} : \left\{ \begin{array}{rcl} \ldots & = & \ldots \\ \mathit{must}^{\uparrow\Sigma'} & = & \mathit{must} \cup (\Sigma' \setminus \Sigma) \end{array} \right.$$

Neutral alphabet extension is by adding

- *may* self-loops for the conjunction and
- *must*+*may* self-loops for the composition

# Modal Interfaces with variable alphabets

We consider Modal Interfaces with consistent states only

$$\begin{array}{rcl} M' \models^{\mathsf{M}} \mathcal{C} & ::= & M' \models^{\mathsf{M}} \mathcal{C}^{\Uparrow\Sigma'} \\ E' \models^{\mathsf{E}} \mathcal{C} & ::= & E' \models^{\mathsf{E}} \mathcal{C}^{\uparrow\Sigma'} \\ \mathcal{C}_1 \wedge \mathcal{C}_2 & ::= & \mathcal{C}_1^{\Uparrow\Sigma} \wedge \mathcal{C}_2^{\Uparrow\Sigma} \\ \mathcal{C}_1 \otimes \mathcal{C}_2 & ::= & \mathcal{C}_1^{\uparrow\Sigma} \otimes \mathcal{C}_2^{\uparrow\Sigma} \\ \mathcal{C}_1 / \mathcal{C}_2 & ::= & \mathcal{C}_1^{\Uparrow\Sigma} / \mathcal{C}_2^{\uparrow\Sigma} \end{array}$$

Neutral alphabet extension is by adding

- *may* self-loops for the conjunction and
- *must*$+$*may* self-loops for the composition

# Bibliographical note

- Modal specifications and Modal automata
  - Hennesy in the 80's
  - Kim Larsen 1987
  - introducing variability in design at low cost
- Revitalized in the late 2000's for use as interface models
  - Kim Larsen, Nyman, Wasowski: modal automata (role of determinism in the complexity of the theory), use for product lines
  - Raclet, Caillaud, Benveniste: modal interfaces, full fledged theory dealing with variable alphabets; correction to a mistake in compatibility

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

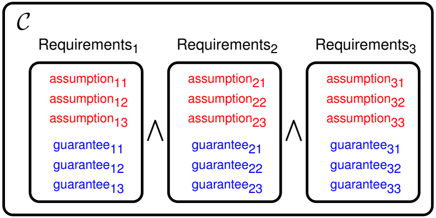
Details

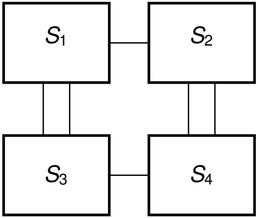
**Contract Based Requirement Engineering**

Details

Concluding Remarks

# Motivations




system architecture (SysML)

mapping $\mathcal{C} = R_1 \wedge R_2 \wedge R_3$
to architecture $S = S_1 \| S_2 \| S_3 \| S_4$
in a best assisted way
to derive $\mathcal{C}_1 \otimes \mathcal{C}_2 \otimes \mathcal{C}_3 \otimes \mathcal{C}_4$
(Benoît Caillaud MICA PoC tool)

# The Parking Garage example

Top-level specification: assumptions & guarantees

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

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

viewpoint **payment**

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

viewpoint **supervisor**

$\mathrm{R}_{\mathrm{g.1}}$(entry)
$\mathrm{R}_{\mathrm{g.1}}$(exit)
$\mathrm{R}_{\mathrm{g.2}}$(entry)
$\mathrm{R}_{\mathrm{g.2}}$(exit)
$\mathrm{R}_{\mathrm{s.1}}$: initially and after !entry_gate close !entry_gate open is forbidden
$\mathrm{R}_{\mathrm{s.2}}$: after !ticket_issued !entry_gate open must be enabled
$\mathrm{R}_{\mathrm{s.3}}$: "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"
$\mathrm{R}_{\mathrm{s.4}}$: "when the entry gate is closed, the entry gate may not open unless a ticket has been issued"
$\mathrm{R}_{\mathrm{s.5}}$: "the entry gate must open when a ticket is issued"
$\mathrm{R}_{\mathrm{s.6}}$: "exit gate must open after an exit ticket is inserted and only then"
$\mathrm{R}_{\mathrm{s.7}}$: "exit gate closes only after vehicle has exited parking"

Each requirement is translated to a Modal Interface
The different Modal Interfaces are combined by using conjunction $\Longrightarrow$ viewpoints
The different viewpoints are combined using conjunction as well

# The Parking Garage example

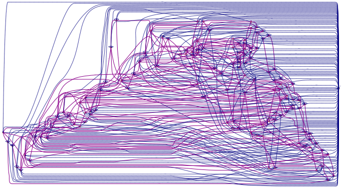
Top-level specification: $\mathcal{C} = \mathcal{C}_{\textbf{entry\_gate}} \wedge \mathcal{C}_{\textbf{exit\_gate}} \wedge \mathcal{C}_{\textbf{payment}} \wedge \mathcal{C}_{\textbf{supervisor}}$

viewpoint **entry_gate** and **exit_gate**
viewpoint **payment**
viewpoint **supervisor**

$\mathrm{R}_{\mathrm{g}.1}$ (entry)
$\mathrm{R}_{\mathrm{g}.1}$ (exit)
$\mathrm{R}_{\mathrm{g}.2}$ (entry)
$\mathrm{R}_{\mathrm{g}.2}$ (exit)
$\mathrm{R}_{\mathrm{s}.1}$: initially and after !entry_gate close !entry_gate open is forbidden
$\mathrm{R}_{\mathrm{s}.2}$: after !ticket_issued !entry_gate open must be enabled
$\mathrm{R}_{\mathrm{s}.3}$: "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"
$\mathrm{R}_{\mathrm{s}.4}$: "when the entry gate is closed, the entry gate may not open unless a ticket has been issued"
$\mathrm{R}_{\mathrm{s}.5}$: "the entry gate must open when a ticket is issued"
$\mathrm{R}_{\mathrm{s}.6}$: "exit gate must open after an exit ticket is inserted and only then"
$\mathrm{R}_{\mathrm{s}.7}$: "exit gate closes only after vehicle has exited parking"

The supervisor as a Modal Interface

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

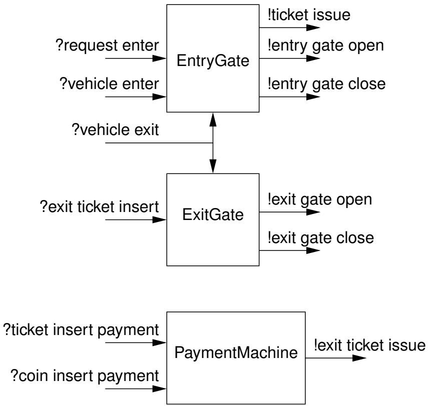


# The Parking Garage example

Architecture for sub-contracting $\mathcal{C}$ as a $\otimes$-composition of sub-systems

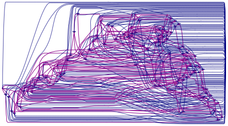
- this is the duty of the designer
- note the change in architecture: supervision performed by the entry gate

$$\mathcal{C} = \left[ \begin{array}{ll} & \mathcal{C}_{\textbf{entry\_gate}} \\ \wedge & \mathcal{C}_{\textbf{exit\_gate}} \\ \wedge & \mathcal{C}_{\textbf{payment}} \\ \wedge & \mathcal{C}_{\textbf{supervisor}} \end{array} \right]$$

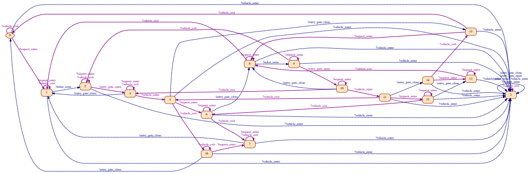


# The Parking Garage example

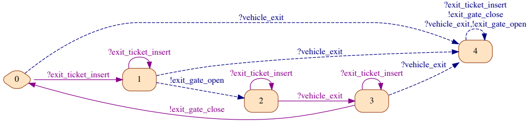
The following $\otimes$-decomposition of $\mathcal{C}$ into three sub-contracts was automatically generated (note the reduction in size)



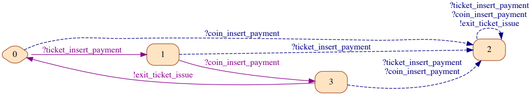
$\mathcal{C}_{\textbf{supervisor}}$



$\mathcal{C}_{\textbf{entry gate}}$




$\mathcal{C}_{\textbf{exit gate}}$




$\mathcal{C}_{\textbf{payment}}$

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

**Details**

Concluding Remarks

# Translating Assumptions and Guarantees into Modal Interfaces

Top-level specification

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

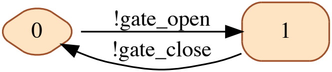
$\mathrm{R}_{g.1}(x)$: "vehicles shall not pass when x_gate is closed",

$\mathrm{R}_{g.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden

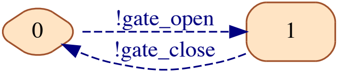
$\mathrm{R}_{g.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the guarantees:

$\mathrm{R}_{g.3}$ as an i/o-automaton:




$\mathrm{R}_{g.3}$ as a Modal Interface:




Note the *may* transitions for outputs

# Translating Assumptions and Guarantees into Modal Interfaces

Top-level specification

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

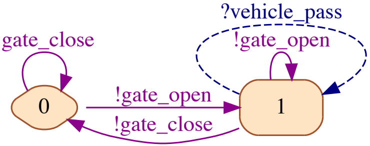
$\mathrm{R}_{\mathrm{g}.1}(x)$: “vehicles shall not pass when x_gate is closed”,

$\mathrm{R}_{\mathrm{g}.2}(x)$: after ?vehicle_pass ?vehicle_pass is forbidden

$\mathrm{R}_{\mathrm{g}.3}$: after !x_gate_open !x_gate_open is forbidden and after !x_gate_close !x_gate_close is forbidden

Translating the assumptions:

$\mathrm{R}_{\mathrm{g}.1}(x)$:




Note the *must* transitions for outputs and the *may* transitions for inputs

Finally the contract for the gate viewpoint is:

$$\mathcal{C}_{\text{gate}} = ((\mathrm{R}_{\mathrm{g}.1} \wedge \mathrm{R}_{\mathrm{g}.2}) \otimes \mathrm{R}_{\mathrm{g}.3})/(\mathrm{R}_{\mathrm{g}.1} \wedge \mathrm{R}_{\mathrm{g}.2})$$

A meta-theory of contracts

Details

Meta-theory $\mapsto$ Assume/Guarantee contracts

Details

Meta-theory $\mapsto$ Interface Automata

Details

Meta-theory $\mapsto$ Modal Interfaces

Details

Contract Based Requirement Engineering

Details

Concluding Remarks

# Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains

# Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual “local reasoning” $\longrightarrow$ system-wide properties

# Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual “local reasoning” $\longrightarrow$ system-wide properties
- Extending the formal scope of contracts:
  - abstractions
  - observers

# Concluding Remarks

- Contracts: large system design by distributed OEM/supplier chains
- Contracts support both formal and semi-formal use:
  - formal: possible for specific contract formalisms
  - semi-formal: manual "local reasoning" $\longrightarrow$ system-wide properties
- Extending the formal scope of contracts:
  - abstractions
  - observers
- The meta-theory clarifies the unique features
  - of contract-based reasoning, versus
  - other techniques of compositional reasoning
- Meta-theory specializes to various formalisms (more than shown)

# More...

- Use of Modal Interfaces for the separate compilation of multiple-clocked synchronous programs, showing that contracts yield useful and non trivial theories of interfaces [Benven. & al. 2012]

- Perspective: meta-theory to support heterogeneity

- Perspective: synchronizing safety with {function+physics}:
  - safety analysis: abstract reliability or fault tree models
  - {function+physics+faults}: too complex for being analysable

  $\Longrightarrow$ for use as observers for safety analysis models