Safety Contracts for Timed Reactive Components

Iulia Dragomir, Iulian Ober and Christian Percebois

IRIT - University of Toulouse

March 21, 2013
Outline

1. Motivation
2. Contract-based Reasoning
3. Component framework: Timed Input/Output Automata
4. A toy example
5. Contract framework for Timed Input/Output Automata
6. Applying contract-based reasoning on the toy example
7. Conclusions
1 Motivation
Context & Problematics

**Context**: development of component-based critical real-time embedded systems

Let $S$ be a component-based system and $\phi_1, \ldots, \phi_n$ a set of requirements.

- A requirement is in general satisfied by the collaboration of a set of components.
- Each component is involved in the satisfaction of several requirements.

⇒ component abstractions
Context: development of component-based critical real-time embedded systems

Let $S$ be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.
**Context & Problematics**

**Context:** development of component-based critical real-time embedded systems

Let $S$ be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

- A requirement is in general satisfied by the collaboration of a set of components
Context & Problematics

Context: development of component-based critical real-time embedded systems

Let $S$ be a component-based system and $\varphi_1, \ldots, \varphi_n$ a set of requirements.

- A requirement is in general satisfied by the collaboration of a set of components
- Each component is involved in the satisfaction of several requirements
Context & Problematics

**Context:** development of component-based critical real-time embedded systems

Let $S$ be a component-based system and $\varphi_1, \cdots, \varphi_n$ a set of requirements.

- A requirement is in general satisfied by the collaboration of a set of components
- Each component is involved in the satisfaction of several requirements
  $\Rightarrow$ the need for components abstractions
Verification by abstractions
Verification by abstractions
Verification by abstractions

Abstraction

\[ \Phi_1 \]
\[ \Phi_2 \]
\[ \Phi_3 \]
\[ \Phi_4 \]
\[ \Phi_5 \]
\[ \Phi_6 \]
\[ \Phi_7 \]

Not sufficient!
Verification by abstractions

Abstraction

Context

!a

!b

?a

?b
Verification by abstractions

Abstraction

Context

!a
!b

?a
?b
Verification by abstractions

Abstraction      Context

!b

α = !b
Verification by abstractions

Deadlock due to the abstraction

Abstraction

Context

α = !b
Deadlock due to the abstraction
→ Not sufficient!
Verification by abstractions

- Deadlock due to the abstraction → Not sufficient!
- An abstraction has to be *correct* in a *context*
Verification by abstractions

- Deadlock due to the abstraction → Not sufficient!
- An abstraction has to be correct in a context → usage of contracts

Abstraction

Context

\( \alpha = !b \)
Introducing contracts

Contract:

- defines partial and abstract component specification for one component and one requirement
- is a pair (assumption, guarantee)
Introducing contracts

Contract:
- defines partial and abstract component specification for one component and one requirement
Introducing contracts

Contract:
- defines partial and abstract component specification for one component and one requirement
- is a pair (assumption, guarantee)
Using contracts: why?

- Requirement specification and decomposition
Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements
Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements
- Model reviews
Using contracts: why?

- Requirement specification and decomposition
- Mapping and tracing requirements
- Model reviews
- Verification of system designs (in SysML)
2 Contract-based Reasoning
Contract-based Reasoning
Contract-based Reasoning

Iulia Dragomir (IRIT)
Contract-based Reasoning

Step 1) Satisfaction
\[ K_i \models C_i, \forall i \]
Contract-based Reasoning

Step 1) Satisfaction
\[ K_i \models C_i \quad \forall i \]
Contract-based Reasoning

Step 1) Satisfaction
\[ K_i \models C_i, \forall i \]

Step 2) Dominance
\{C_1, \ldots, C_n\} dominates C
Contract-based Reasoning

Step 1) Satisfaction
\[ \text{K}_i \models \text{C}_i, \forall i \]

Step 2) Dominance
\{\text{C}_1, \ldots, \text{C}_n\} \text{ dominates C}

Step 3) Conformance
\[ \text{A} \parallel \text{G} \leq \varphi \]
Contract-based approach for component-based systems

- Formalization of component framework
Contract-based approach for component-based systems

- Formalization of component framework
- Verification relations
Contract-based approach for component-based systems

- Formalization of component framework
- Verification relations
  1. Contract satisfaction
  2. Dominance between contracts
  3. Conformance
Component framework: Timed Input/Output Automata
Definition

Timed input/output automaton $A$
Timed input/output automaton \( A = (X, \ldots) \)
Definition

Timed input/output automaton $A = (X, Clk,$
Timed input/output automaton $\mathcal{A} = (X, \text{Clk}, Q, $
Definition

Timed input/output automaton $A = (X, Clk, Q, \theta, \theta)$
Definition

Timed input/output automaton $A = (X, Clk, Q, \theta, I, \ldots)$
Component: Timed Input/Output Automaton

**Definition**

Timed input/output automaton $A = (X, \text{Clk}, Q, \theta, I, O,$

![Diagram of a timed input/output automaton with states and transitions labeled with $b$, $q$, $i:=0$, $?b/q$, $!a/x$, $?b/q$.](image)
**Component: Timed Input/Output Automaton**

**Definition**

Timed input/output automaton \( A = (X, Clk, Q, \theta, I, O, V, \) \)
Timed input/output automaton $A = (X, Clk, Q, \theta, I, O, V, H,$
Component: Timed Input/Output Automaton

Definition

Timed input/output automaton \( A = (X, Clk, Q, \theta, I, O, V, H, D, \) \)
Component: Timed Input/Output Automaton

Definition

Timed input/output automaton \( A = (X, Clk, Q, \theta, I, O, V, H, D, T) \)
Properties of a timed input/output automaton

1. Existence of point trajectories: \( \forall x \in Q, \gamma(x) : [0, 0] \rightarrow x \in T \)
Properties of a timed input/output automaton

1. Existence of point trajectories: \( \forall x \in Q, \gamma(x) : [0, 0] \rightarrow x \in \mathcal{T} \)

2. Prefix, suffix and concatenation closure of \( \mathcal{T} \)
Properties of a timed input/output automaton

1. Existence of point trajectories: \( \forall x \in Q, \gamma(x) : [0, 0] \rightarrow x \in \mathcal{T} \)

2. Prefix, suffix and concatenation closure of \( \mathcal{T} \)

3. Input actions enabling: \( \forall x \in Q, \forall a \in I, \exists x' \in Q \) such that \( x \xrightarrow{?a} x' \)
Properties of a timed input/output automaton

1. Existence of point trajectories: \( \forall x \in Q, \gamma(x) : [0, 0] \to x \in T \)

2. Prefix, suffix and concatenation closure of \( T \)

3. Input actions enabling: \( \forall x \in Q, \forall a \in I, \exists x' \in Q \text{ such that } x \xrightarrow{a} x' \)

4. Time-passage enabling: \( \forall x \in Q, \exists \tau \in T \text{ such that } \tau(0) = x \) and either
   - \( \tau.\text{limit\_time} = \infty \) or
   - \( \tau \) is closed and some \( l \in O \cup V \cup H \) is enabled is \( \tau(\tau.\text{limit\_time}) \)
TIOA behaviour

- **Execution fragment**: sequence of trajectories and actions
  
  Example:
**Execution fragment**: sequence of trajectories and actions

Example: \( \alpha = [0, 0] \)
**Execution fragment**: sequence of trajectories and actions

Example: $\alpha = [0, 0]!a$
**TIOA behaviour**

*Execution fragment*: sequence of trajectories and actions

Example: $\alpha = [0, 0]!a[0, \delta]$
TIOA behaviour

**Execution fragment**: sequence of trajectories and actions

Example: \( \alpha = [0, 0]!a[0, \delta]\epsilon \)
TIOA behaviour

- **Execution fragment**: sequence of trajectories and actions

Example: \( \alpha = [0, 0]!a[0, \delta] \epsilon [0, \delta] \)
Execution fragment: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b$
**Execution fragment**: sequence of trajectories and actions

Example: $\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]$
**Execution fragment**: sequence of trajectories and actions

Example: \( \alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c \)
Execution fragment: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0] \)
Execution fragment: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0] \downarrow b$
**Execution fragment**: sequence of trajectories and actions

Example: $\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$
Execution fragment: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

Trace: sequence of time-passage lengths and external actions
Example: $\text{trace}(\alpha) =$
Execution fragment: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta] \epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0] \)

Trace: sequence of time-passage lengths and external actions
Example: \( \text{trace}(\alpha) = [0, 0] \)
**Execution fragment**: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

**Trace**: sequence of time-passage lengths and external actions
Example: $\text{trace}(\alpha) = [0, 0]!a$
Execution fragment: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0] \downarrow b[0, 0] \)

Trace: sequence of time-passage lengths and external actions
Example: \( trace(\alpha) = [0, 0]!a[0, \delta] \)
**Execution fragment**: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta] e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

**Trace**: sequence of time-passage lengths and external actions
Example: $\text{trace}(\alpha) = [0, 0]!a[0, \delta]$
Execution fragment: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

Trace: sequence of time-passage lengths and external actions
Example: $\text{trace}(\alpha) = [0, 0]!a[0, \delta][0, \delta]$
Execution fragment: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

Trace: sequence of time-passage lengths and external actions
Example: $\text{trace}(\alpha) = [0, 0]!a[0, \delta][0, \delta]$
**Execution fragment**: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0] \)

**Trace**: sequence of time-passage lengths and external actions
Example: \( \text{trace}(\alpha) = [0, 0]!a[0, 2*\delta] \)
**Execution fragment**: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta] \epsilon [0, \delta] ?b [0, 0] c[0, 0] \downarrow b[0, 0] \)

**Trace**: sequence of time-passage lengths and external actions
Example: \( trace(\alpha) = [0, 0]!a[0, 2 \cdot \delta] ?b \)
**Execution fragment**: sequence of trajectories and actions
Example: $\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

**Trace**: sequence of time-passage lengths and external actions
Example: $trace(\alpha) = [0, 0]!a[0, 2\times \delta]?b[0, 0]$
**Execution fragment**: sequence of trajectories and actions

Example: $$\alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$$

**Trace**: sequence of time-passage lengths and external actions

Example: $$\text{trace}(\alpha) = [0, 0]!a[0, 2 \times \delta]?b[0, 0]c$$
TIOA behaviour

- **Execution fragment**: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!_a[0, \delta]_e[0, \delta]?_b[0, 0]c[0, 0]\downarrow_b[0, 0]$

- **Trace**: sequence of time-passage lengths and external actions
  Example: $\text{trace}(\alpha) = [0, 0]!_a[0, 2 \ast \delta]?_b[0, 0]c[0, 0]$
TIOA behaviour

- **Execution fragment**: sequence of trajectories and actions
  Example: \( \alpha = [0, 0]!a[0, \delta] \epsilon[0, \delta]?b[0, 0]c[0, 0] \downarrow b[0, 0] \)

- **Trace**: sequence of time-passage lengths and external actions
  Example: \( trace(\alpha) = [0, 0]!a[0, 2*\delta]?b[0, 0]c[0, 0] \)
**Execution fragment**: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta]\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0] \)

**Trace**: sequence of time-passage lengths and external actions
Example: \( trace(\alpha) = [0, 0]!a[0, 2*\delta]?b[0, 0]c[0, 0][0, 0] \)
TIOA behaviour

- **Execution fragment**: sequence of trajectories and actions
  Example: $\alpha = [0, 0]!a[0, \delta]_\epsilon[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0]$

- **Trace**: sequence of time-passage lengths and external actions
  Example: $\text{trace}(\alpha) = [0, 0]!a[0, 2 \times \delta]?b[0, 0]c[0, 0][0, 0]$

\[\text{Diagram}\]
**Execution fragment**: sequence of trajectories and actions
Example: \( \alpha = [0, 0]!a[0, \delta]e[0, \delta]?b[0, 0]c[0, 0]\downarrow b[0, 0] \)

**Trace**: sequence of time-passage lengths and external actions
Example: \( trace(\alpha) = [0, 0]!a[0, 2 \ast \delta]?b[0, 0]c[0, 0] \)
Composition compatibility:

\[ Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset, \text{ for } i \neq j \]
Composition compatibility:
\[ Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset, \text{ for } i \neq j \]

Parallel composition:
\[
\begin{align*}
\frac{x_1 \xrightarrow{a} x'_1}{(x_1 \cup x_2) \xrightarrow{a} (x'_1 \cup x_2)} (a \in A_1 \setminus A_2) \\
\frac{x_2 \xrightarrow{a} x'_2}{(x_1 \cup x_2) \xrightarrow{a} (x_1 \cup x'_2)} (a \in A_2 \setminus A_1)
\end{align*}
\]
Composition compatibility:
\[ Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset, \text{ for } i \neq j \]

Parallel composition:
\[
\begin{align*}
  &x_1 \xrightarrow{a} x'_1 \\
  &\quad \quad (x_1 \cup x_2) \xrightarrow{a} (x'_1 \cup x_2) \\
  &\quad \quad (a \in A_1 \setminus A_2) \\
  &\quad \quad (x_1 \cup x_2) \xrightarrow{a} (x_1 \cup x'_2) \\
  &\quad \quad (a \in A_2 \setminus A_1) \\
  &x_1 \xrightarrow{a} x'_1 \land x_2 \xrightarrow{a} x'_2 \\
  &\quad \quad (x_1 \cup x_2) \xrightarrow{a} (x'_1 \cup x'_2) \\
  &\quad \quad (a \in (A_1 \cap A_2) \cup (T_1 \land T_2))
\end{align*}
\]
TIOA composition

Composition compatibility:
\[ Y_i \cap Y_j = H_i \cap A_j = V_i \cap A_j = O_i \cap O_j = I_i \cap I_j = \emptyset, \text{ for } i \neq j \]

Parallel composition:

\[
\begin{align*}
    x_1 & \xrightarrow{a} x'_1 \\
    (x_1 \cup x_2) & \xrightarrow{a} (x'_1 \cup x_2) \\
    x_2 & \xrightarrow{a} x'_2 \\
    (x_1 \cup x_2) & \xrightarrow{a} (x'_1 \cup x'_2) \\
    x_1 & \xrightarrow{a} x'_1 \land x_2 & \xrightarrow{a} x'_2 \\
    (x_1 \cup x_2) & \xrightarrow{a} (x'_1 \cup x'_2) \\
\end{align*}
\]

\(a \in A_1 \setminus A_2\)

\(a \in A_2 \setminus A_1\)

\(a \in (A_1 \cap A_2) \cup (T_1 \land T_2)\)

Theorem

The parallel composition operator is commutative and associative.
Outline

4 A toy example
Running example
Property \( \varphi \) to be checked

Given \( \delta_1 < \delta_2 \), the subsystem doesn’t emit consecutive \( a \)'s or \( b \)'s.
5 Contract framework for Timed Input/Output Automata
**Formal contract**

*Component K*: a timed input/output automaton
Formal contract

Component $K$: a timed input/output automaton

Closed component: $I = O = \emptyset$
Open component: it is not closed
**Formal contract**

*Component K*: a timed input/output automaton

*Closed component*: \( I = O = \emptyset \)

*Open component*: it is not closed

*Environment E* for \( K \): a timed input/output automaton compatible with \( K \) such that \( I_E \subseteq O_K \) and \( O_E \subseteq I_K \)
Formal contract

Component $K$: a timed input/output automaton

Closed component: $I = O = \emptyset$

Open component: it is not closed

Environment $E$ for $K$: a timed input/output automaton compatible with $K$ such that $I_E \subseteq O_K$ and $O_E \subseteq I_K$

Definition

A contract for a component $K$ is a pair $(A, G)$ of TIOA such that $I_A = O_G$ and $O_A = I_G$ (i.e. the composition is a closed system) and $I_G \subseteq I_K$ and $O_G \subseteq O_K$ (i.e. the interface of $K$ is a refinement of that of $G$).
Contracts for the running example

$C_1 = (A_1, G_1)$ contract for $K_1$

$A_1$

$G_1$
Contracts for the running example

\[ C_1 = (A_1, G_1) \text{ contract for } K_1 \]

\[ C_2 = (A_2, G_2) \text{ contract for } K_2 \]
Contracts for the running example

$C_1 = (A_1, G_1)$ contract for $K_1$

$C_2 = (A_2, G_2)$ contract for $K_2$

$C_3 = (A_3, G_3)$ contract for $K_3$
Conformance relation

Definition

Let $K_1$ and $K_2$ be two comparable components (i.e. having the same external interface).

$K_1 \preceq K_2$ if $\text{traces}(K_1) \subseteq \text{traces}(K_2)$.

Theorem

Conformance is a preorder relation.

Theorem

Let $K_1$ and $K_2$ be two comparable components with $K_1 \preceq K_2$ and $E$ a component compatible with both $K_1$ and $K_2$. Then $K_1 \parallel E \preceq K_2 \parallel E$. 
Conformance relation

**Definition**

Let $K_1$ and $K_2$ be two comparable components (i.e. having the same external interface). $K_1 \preccurlyeq K_2$ if $\text{traces}_{K_1} \subseteq \text{traces}_{K_2}$.
Conformance relation

**Definition**

Let $K_1$ and $K_2$ be two comparable components (i.e. having the same external interface). $K_1 \preceq K_2$ if $\text{traces}_{K_1} \subseteq \text{traces}_{K_2}$.

**Theorem**

Conformance is a preorder relation.
Definition
Let $K_1$ and $K_2$ be two comparable components (i.e. having the same external interface). $K_1 \preceq K_2$ if $\text{traces}_{K_1} \subseteq \text{traces}_{K_2}$.

Theorem
Conformance is a preorder relation.

Theorem
Let $K_1$ and $K_2$ be two comparable components with $K_1 \preceq K_2$ and $E$ a component compatible with both $K_1$ and $K_2$. Then $K_1 \parallel E \preceq K_2 \parallel E$. 
Refinement under context relation

**Definition**

Let $K_1$ and $K_2$ be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$, and $V_{K_2} \subseteq V_{K_1}$. Let $E$ be an environment for $K_1$ compatible with both $K_1$ and $K_2$. We say that $K_1$ refines $K_2$ in the context of $E$, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where $K'$ and $E'$ are defined such that both members of the conformance relation are comparable and closed.
Refinement under context relation

Definition

Let $K_1$ and $K_2$ be two components such that $I_{K_2} \subseteq I_{K_1} \cup V_{K_1}$, $O_{K_2} \subseteq O_{K_1} \cup V_{K_1}$ and $V_{K_2} \subseteq V_{K_1}$. Let $E$ be an environment for $K_1$ compatible with both $K_1$ and $K_2$. We say that $K_1$ refines $K_2$ in the context of $E$, denoted $K_1 \sqsubseteq_E K_2$, if

$$K_1 \parallel E \parallel E' \preceq K_2 \parallel E \parallel K' \parallel E'$$

where $K'$ and $E'$ are defined such that both members of the conformance relation are comparable and closed.

Definition

$$K \models C = (A, G) \iff K \sqsubseteq_A G$$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Example: $K_1 \sqsubseteq_{A_1} G_1$
Properties of refinement under context

**Theorem**
Given a set $\mathcal{K}$ of comparable components and a fixed environment $E$ for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

Let $K_1$ and $K_2$ be two components and $E$ an environment compatible with both $K_1$ and $K_2$ such that $E = E_1 \parallel E_2$.

$K_1 \sqsubseteq E_1 \parallel E_2 K_2$ if and only if $K_1 \parallel E_1 \sqsubseteq E_2 K_2 \parallel E_1$.

**Theorem**
Let $K$ be a component, $E$ its environment and $C = (A, G)$ the contract for $K$ such that $K$ and $G$ are compatible with $E$ and $A$. If

1. traces $G$ is closed under limits,
2. traces $G$ is closed under time-extension,
3. $K \sqsubseteq A G$ and
4. $E \sqsubseteq G A$

then $K \sqsubseteq E G$.
Properties of refinement under context

Theorem

Given a set \( \mathcal{K} \) of comparable components and a fixed environment \( E \) for that interface, the refinement under context relation \( \sqsubseteq_E \) is a preorder over \( \mathcal{K} \).

Theorem

Let \( K_1 \) and \( K_2 \) be two components and \( E \) an environment compatible with both \( K_1 \) and \( K_2 \) such that \( E = E_1 \parallel E_2 \).

\[
K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \iff K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1
\]
Properties of refinement under context

**Theorem**

Given a set $\mathcal{K}$ of comparable components and a fixed environment $E$ for that interface, the refinement under context relation $\sqsubseteq_E$ is a preorder over $\mathcal{K}$.

**Theorem**

Let $K_1$ and $K_2$ be two components and $E$ an environment compatible with both $K_1$ and $K_2$ such that $E = E_1 \parallel E_2$.

$$K_1 \sqsubseteq_{E_1 \parallel E_2} K_2 \iff K_1 \parallel E_1 \sqsubseteq_{E_2} K_2 \parallel E_1$$

**Theorem**

Let $K$ be a component, $E$ its environment and $C = (A, G)$ the contract for $K$ such that $K$ and $G$ are compatible with each of $E$ and $A$. If (1) $\text{traces}_G$ is closed under limits, (2) $\text{traces}_G$ is closed under time-extension, (3) $K \sqsubseteq_A G$ and (4) $E \sqsubseteq_G A$ then $K \sqsubseteq_E G$. 
Abstract system

\[ K \]

\[ K_1 \]

- \( i := 0 \)
- \( m \)
- \( \downarrow q \)
- \( !n(i) \)
- \( \downarrow m \)
- \( !n(i) \)
- \( \downarrow q \)
- \( !m / i := i + 1 \)
- \( !a / i := i + 1 \)

\[ [x = \delta_1] \]
\[ !p / x := 0 \]

- \( p \)
- \( q \)

\[ K_2 \]

- \( j := 0 \)
- \( \downarrow u \)
- \( !v(j) \)
- \( \downarrow p \)
- \( !b / y := 0; j := j + 1 \)
- \( !b / j := j + 1 \)

\[ K_3 \]

- \( \downarrow n \)
- \( !m \)
- \( \downarrow v \)
- \( !u \)

Iulia Dragomir (IRIT)
Abstract system

\[ G_1 \]
\[ G_2 \]
\[ G_3 \]
Top contract for the abstract system

A

\[ a \rightarrow \downarrow b \rightarrow \downarrow a \text{ eager} \]

G

\[ [z = \delta_2] \rightarrow !b / z := 0 \rightarrow \text{ lazy} \rightarrow \text{ eager} \rightarrow a \rightarrow b \]
Definition

\[ \{ C_i \}_{i=1}^n \ \text{dominates} \ C \ \text{iff} \ \forall \{ K_i \}_{i=1}^n \ \text{such that}, \ \forall i, \ K_i \models C_i, \ \text{we have} \]
\[ (K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C. \]
Contract dominance

Definition
\[
\{C_i\}_{i=1}^n \text{ dominates } C \text{ iff } \forall \{K_i\}_{i=1}^n \text{ such that, } \forall i, K_i \models C_i, \text{ we have }
(K_1 \parallel K_2 \parallel \cdots \parallel K_n) \models C.
\]

Theorem
\[
\{C_i\}_{i=1}^n \text{ dominates } C \text{ if, } \forall i, \text{ traces}_{A_i}, \text{ traces}_{G_i}, \text{ traces}_A \text{ and traces}_G \text{ are closed under limits and under time-extension and}
\begin{align*}
G_1 \parallel \cdots \parallel G_n & \sqsubseteq_A G \\
A \parallel G_1 \parallel \cdots \parallel G_{i-1} \parallel G_{i+1} \parallel \cdots \parallel G_n & \sqsubseteq_{G_i} A_i, \forall i
\end{align*}
\]
Applying contract-based reasoning on the toy example
Verifying Step 1

Step 1) Satisfaction
\[ K_i \models C_i, \forall i \]

Step 2) Dominance
\{C_1, ..., C_n\} dominates C

Step 3) Conformance
\[ A \parallel G \leq \varphi \]
Verifying Step 1

1. $K_1 \models C_1$
2. $K_2 \models C_2$
3. $K_3 \models C_3$
Verifying Step 2

Step 3) Conformance

$A \parallel G \leq \varphi$

Step 2) Dominance

$\{C_1, ..., C_n\}$ dominates $C$

Step 1) Satisfaction

$K_i \models C_i, \forall i$
Verifying Step 2

\[ \{ C_1, C_2, C_3 \} \text{ dominates } C: \]
Verifying Step 2

\{C_1, C_2, C_3\} dominates \(C\):

1. \(\text{traces}_A, \text{traces}_{A_1}, \text{traces}_{A_2}, \text{traces}_{A_3}\) are closed under limits and under time-extension

2. \(\text{traces}_G, \text{traces}_{G_1}, \text{traces}_{G_2}, \text{traces}_{G_3}\) are closed under limits and under time-extension
Verifying Step 2

\{ C_1, C_2, C_3 \} \text{ dominates } C:

1. \text{traces}_A, \text{traces}_{A_1}, \text{traces}_{A_2}, \text{traces}_{A_3} \text{ are closed under limits and under time-extension}

2. \text{traces}_G, \text{traces}_{G_1}, \text{traces}_{G_2}, \text{traces}_{G_3} \text{ are closed under limits and under time-extension}

3. G_1 \parallel G_2 \sqsubseteq_A G
Verifying Step 2

\{ C_1, C_2, C_3 \} \text{ dominates } C:

1. \( \text{traces}_A, \text{traces}_{A_1}, \text{traces}_{A_2}, \text{traces}_{A_3} \) are closed under limits and under time-extension

2. \( \text{traces}_G, \text{traces}_{G_1}, \text{traces}_{G_2}, \text{traces}_{G_3} \) are closed under limits and under time-extension

3. \( G_1 \parallel G_2 \sqsubseteq_A G \)

4. \( A \parallel G_2 \sqsubseteq_{G_1} A_1 \)

5. \( A \parallel G_1 \sqsubseteq_{G_2} A_2 \)

6. \( A \parallel G_1 \parallel G_2 \sqsubseteq_{G_3} A_3 \)
Verifying Step 3

Step 1) Satisfaction

\( K_i \models C_i, \forall i \)

Step 2) Dominance

\( \{C_1, \ldots, C_n\} \) dominates \( C \)

Step 3) Conformance

\( A \parallel G \leq \varphi \)
Verifying Step 3
Conclusions
Results

- TIOA component framework
Results

- TIOA component framework
- Formal contract with interface refinement
Results

- TIOA component framework
- Formal contract with interface refinement
- Refinement relations based on trace inclusion
Results

- TIOA component framework
- Formal contract with interface refinement
- Refinement relations based on *trace inclusion*
- Applied on a toy example
Related work

- (Timed) Interface Automata
- Interface Input/Output Automata
- Timed Input/Output Automata in ECDAR
Future work

Contract framework for TIOA
Future work

Contract framework for TIOA

Automatic checking of verification relations
Future work

- Contract-based theory for SysML
- Contract framework for TIOA
- Automatic checking of verification relations
Future work

- Contract-based theory for SysML
- Contract framework for TIOA
- Automatic checking of verification relations

+ Comparative efficiency study
How to build contracts?
How to build contracts?

- Solve for $G$
1. How to build contracts?
   - Solve for $G$
   - Automatically generate $A$
How to build contracts?

1. Solve for $G$
2. Automatically generate $A$

Automation and integration within a development process
References:


Thank you!

Any questions?