Modal Interfaces:

Unifying Interface Automata and Modal Specifications

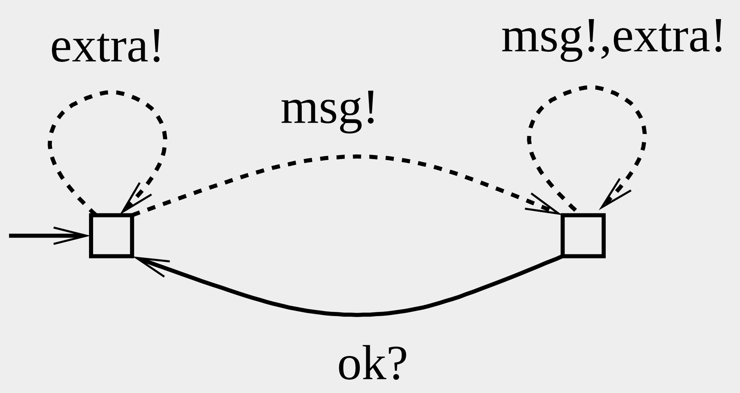
J.-B.Raclet[1], E.Badouel[1], A.Benveniste[1], B.Caillaud[1], A.Legay[1] and R.Passerone[2]

[1] INRIA, France, {*name.surname@inria.fr*} [2] University of Trento, Italy, *roberto.passerone@unitn.it*

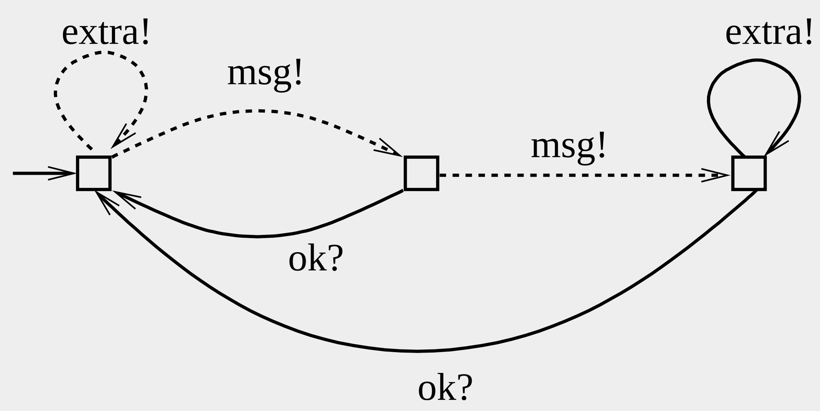
1 Semantic of Modal Interfaces

The modal interface theory is a unification of:

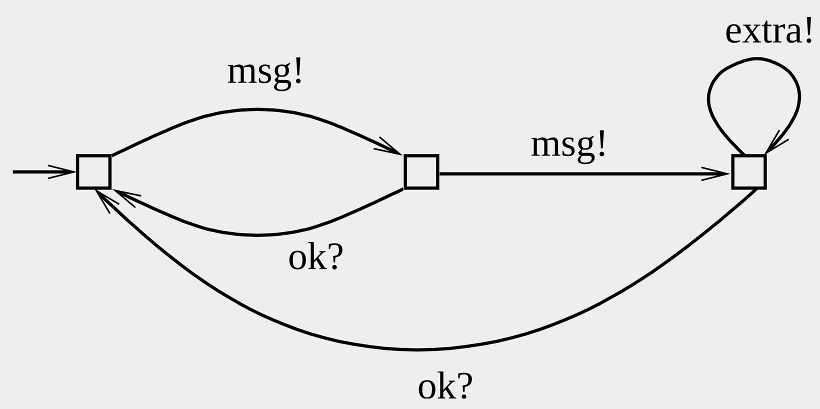
- **Modal specifications**: transitions are given a modality **may** or/and **must** with consistency $must \subseteq may$;
- **Interface automata**: actions are originated either from the component (**output**: $a!$) or from the environment (**input**: $a?$); input-enabledness is not required.



Refinement $\leq$: have more must and less may.



Implementation $\models$: keep all must and some may.



Refinement entails substituability as it is sound and complete:

$$\mathcal{S}_1 \leq \mathcal{S}_2 \;\Leftrightarrow\; Implem(\mathcal{S}_1) \subseteq Implem(\mathcal{S}_2)$$

2 Conjunction

Conjunction is needed for:

- Composition of **different viewpoints** for a same component;
- **Shared refinement**: use an implementation for several interfaces.

Product of underlying automata; $\cap$ may, $\cup$ must :

$\wedge$	$\rightarrow$	$\dashrightarrow$	$\nrightarrow$
$\rightarrow$	$\rightarrow$	$\rightarrow$	$\lightning$
$\dashrightarrow$	$\rightarrow$	$\dashrightarrow$	$\nrightarrow$
$\nrightarrow$	$\lightning$	$\nrightarrow$	$\nrightarrow$

and backward pruning of inconsistencies $\lightning$.

$\mathcal{S}_1 \wedge \mathcal{S}_2$ is the **GLB** of $\mathcal{S}_1$ and $\mathcal{S}_2$; moreover:

$$Implem(\mathcal{S}_1 \wedge \mathcal{S}_2) = Implem(\mathcal{S}_1) \cap Implem(\mathcal{S}_2)$$

3 Product

The product reflects the standard composition of implementations.

Product of underlying automata; $\cap$ may, $\cap$ must :

$\otimes$	$\rightarrow$	$\dashrightarrow$	$\nrightarrow$
$\rightarrow$	$\rightarrow$	$\dashrightarrow$	$\nrightarrow$
$\dashrightarrow$	$\dashrightarrow$	$\dashrightarrow$	$\nrightarrow$
$\nrightarrow$	$\nrightarrow$	$\nrightarrow$	$\nrightarrow$

The product is associative. Moreover it ensures **independent design**:

- $\mathcal{S}'_1 \leq \mathcal{S}_1$ and $\mathcal{S}'_2 \leq \mathcal{S}_2 \;\Rightarrow\; \mathcal{S}'_1 \otimes \mathcal{S}'_2 \leq \mathcal{S}_1 \otimes \mathcal{S}_2$;
- $\mathcal{I}_1 \models \mathcal{S}_1$ and $\mathcal{I}_2 \models \mathcal{S}_2 \;\Rightarrow\; \mathcal{I}_1 \otimes \mathcal{I}_2 \models \mathcal{S}_1 \otimes \mathcal{S}_2$.

4 Quotient

Quotient is needed for:

- Solving **equations**, $max_{\mathcal{X}} : \mathcal{S}_1 \otimes \mathcal{X} \leq \mathcal{S}$;
- Implication for **Assume/Guarantee contracts**: $\mathcal{G}/\mathcal{A}$.

Product of underlying automata; may/must as follows:

$/$	$\rightarrow$	$\dashrightarrow$	$\nrightarrow$
$\rightarrow$	$\rightarrow$	$\lightning$	$\lightning$
$\dashrightarrow$	$\dashrightarrow$	$\dashrightarrow$	$\dashrightarrow$
$\nrightarrow$	$\nrightarrow$	$\nrightarrow$	$\dashrightarrow$

and backward pruning of inconsistencies $\lightning$.

- $\mathcal{S}_1 \otimes \mathcal{S}_2 \leq \mathcal{S} \;\Leftrightarrow\; \mathcal{S}_2 \leq \mathcal{S}/\mathcal{S}_1$;
- $\mathcal{I}_2 \models \mathcal{S}/\mathcal{S}_1 \;\Leftrightarrow\; \forall \mathcal{I}_1 \models \mathcal{S}_1, \mathcal{I}_1 \otimes \mathcal{I}_2 \models \mathcal{S}$.

5 Dealing with dissimilar alphabets

Needed as implementations can have a larger alphabet and interfaces may have different alphabets.

Alphabet equalization should be **neutral**: it should not constrain what other interfaces may want to require regarding these extra actions:

- For $\mathcal{S}_1 \wedge \mathcal{S}_2$ equalize by adding **may** self-loops onto $\mathcal{S}_1$ and $\mathcal{S}_2$;
- For $\mathcal{S}_1 \otimes \mathcal{S}_2$ equalize by adding **must** self-loops onto $\mathcal{S}_1$ and $\mathcal{S}_2$;
- For $\mathcal{S}_1/\mathcal{S}_2$ equalize by adding **may** self-loops onto $\mathcal{S}_1$ and **must** self-loops onto $\mathcal{S}_2$.

6 Compatibility

Two interfaces are **compatible** if there exists an environment where they can avoid illegal states (i.e., states where an interface wishes to produce an output that is not accepted as input by the other interface).

Composition $\mathcal{S}_1 \parallel \mathcal{S}_2$ is obtained by:

- computing illegal states $Illegal(\mathcal{S}_1, \mathcal{S}_2)$;
- computing exception states $\mathcal{E} = pre_!(Illegal(\mathcal{S}_1, \mathcal{S}_2))$;
- replacing transitions leading to $\mathcal{E}$ by transitions to a new universal state.

Compatibility is **preserved by refinement** and $\parallel$ is associative and monotonic for the refinement preorder.

9th International Conference on Embedded Software (EMSOFT'09), Embedded Systems Week. Oct. 11-16, 2009. Grenoble, France.