Modal Interfaces: Unifying Interface Automata and Modal Specifications*

Jean-Baptiste Raclet
INRIA Grenoble, France
raclet@inrialpes.fr

Eric Badouel
INRIA/IRISA Rennes, France
eric.badouel@irisa.fr

Albert Benveniste
INRIA/IRISA Rennes, France
albert.benveniste@irisa.fr

Benoît Caillaud
INRIA/IRISA Rennes, France
benoit.caillaud@irisa.fr

Axel Legay
INRIA/IRISA Rennes, France
alegay@irisa.fr

Roberto Passerone
PARADES Scarl, Rome, Italy
rpasserone@parades.rm.cnr.it

ABSTRACT

This paper presents a unification of *interface automata* and *modal specifications*, two radically dissimilar models for interface theories. Interface automata is a game-based model, which allows to make assumptions on the environment and propose an optimistic view for composition : *two components can be composed if there is an environment where they can work together*. Modal specification is a language theoretic account of a fragment of the modal mu-calculus logic that is more complete but which does not allow to distinguish between the environment and the component. Partial unifications of these two frameworks have been explored recently. A first attempt by Larsen et al. considers *modal interfaces*, an extension of modal specifications that deals with compatibility issues in the composition operator. However, this composition operator is incorrect. A second attempt by Raclet et al. gives a different perspective, and emphasises on conjunction and residuation of modal specifications, including when interfaces have dissimilar alphabets, but disregards interface compatibility. The present paper contributes a thorougher unification of the two theories by correcting the modal interface composition operator presented in the paper by Larsen et al., drawing a complete picture of the modal interface algebra, and pushing even further the comparison between interface automata, modal automata and modal interfaces.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*

General Terms

Design, Theory

Keywords

Interface Automata, Modal Specifications, Compatibility

*This work was funded in part by the European IP-SPEEDS project number 033471 and the European STREP-COMBEST project number 215543.

EMSOFT'09, October 12–16, 2009, Grenoble, France.

1. INTRODUCTION

Nowadays, systems are tremendously big and complex, resulting from the assembling of several components. These many components are in general designed by teams, working independently but with a common agreement on what the interface of each component should be. As a consequence, mathematical foundations that allow to reason at the abstract level of interfaces is a very active research area. According to our understanding of industrial needs, an interface theory is at least subject to the following requirements:

1. *Satisfaction and satisfiability are decidable.* Interfaces should be seen as specifications whose models are its possible implementations. It should thus be decidable whether an interface admits an implementation and whether a given component implements a given interface.

2. *Refinement entails substituability.* Refinement allows one to replace, in any context, an interface by a more detailed version of it. Refinement should entail substituability of interface implementations, meaning that every implementation satisfying a refinement also satisfies the larger interface. For the sake of controlling design complexity, it is desirable to be able to decide whether there exists an implementation satisfying two different interfaces. This is called *shared refinement*.

3. *Encompassing interfaces with dissimilar alphabets.* Complex systems are built by combining subsystems possessing dissimilar alphabets for referencing ports and variables. It is thus important to properly handle those different alphabets when combining interfaces.

4. *Composition supports independent design.* The interface theory should also provide a combination operator on interfaces, reflecting the standard composition of implementations by, *e.g.* parallel product. This operation must be associative and commutative to guarantee independence in the development. Depending on

the model, a notion of compatibility for composition may also be considered, i.e., there can be cases where two systems cannot be composed.

5. *Interfaces are closed under conjunction.* It is the current practice that early requirements capture relies on Doors Databases, or even Excel files containing possibly many textual requirements. Under the current practice, little formal support exists to handle them. Moving ahead can be envisioned by formalizing the notation used for individual requirements. This can be, e.g., achieved by relying on so-called semi-formal languages [7], whose sentences are translatable into predefined behavioral patterns according to several viewpoints. Alternatively, graphical scenario languages could be considered [10, 21]. Composing viewpoints within a given subsystem calls for the support of the concept of *conjunction* of interfaces in order to combine requirements and check their satisfiability.

6. *Interface quotient supports incremental design and component reuse.* Last but not least, a quotienting operation, dual to composition is crucial to perform incremental design. Consider a desired global specification and the specification of a preexisting component; the quotient specification describes the part of the global specification that remains to be implemented.

Building good interface theories has been the subject of intensive studies (see e.g., [20, 14, 5, 17, 19, 12, 15]). In this paper we will concentrate on two models: (1) *interface automata* [14] and (2) *modal specifications* [22]. Interface automata is a game semantics based variation of input/output automata which deals with open systems, their refinement and composition, and put the emphasis on interface compatibility. Modal specifications is a language theoretic account of a fragment of the modal mu-calculus logic [18] which admits a richer composition algebra with product, conjunction, and residuation operators.

In interface automata [14], an interface is represented by an input/output automaton [26], *i.e.*, an automaton whose transitions are labeled with *input* or *output* actions. The semantics of such an automaton is given by a two-player game: an *Input* player represents the environment, and an *Output* player represents the component itself. Interface automata do not encompass any notion of model, because one cannot distinguish between interfaces and implementations. Alternatively, properties of interfaces are described in game-based logics, *e.g.*, ATL [1], with a high-cost complexity.

Refinement between interface automata corresponds to the alternating refinement relation between games [2], *i.e.*, an interface refines another one if its environment is more permissive whereas its component is more restrictive. Shared refinement is defined in an ad-hoc manner [16] for a particular class of interfaces [8]. Contrary to most interfaces theories, the game-based interpretation offers an *optimistic* treatment of composition: two interfaces can be composed if there exists at least one environment (i.e., one strategy for the Input player) in which they can interact together in a safe way (i.e., whatever the strategy of the Output player is). This is referred as compatibility of interfaces.

Modal specifications [22] correspond to *deterministic modal automata*, i.e., automata whose transitions are typed with *may* and *must* modalities. A modal specification thus represents a set of models; informally, a must transition is available in every component that implements the modal specification, while a may transition needs not be. The components that implement modal specifications are prefix-closed languages, or equivalently deterministic automata.

Satisfiability of modal specifications is decidable. Refinement between modal specifications coincides with models inclusion. Since components can be seen as specifications where all transitions are typed must (all possible implementation choices have been made), satisfaction is also expressed via alternating simulation. Conjunction is effectively computed via a product-like construction. Combination of modal specifications, handling synchronization products *à la* Arnold and Nivat [3], and the dual quotient combinators can be efficiently handled in this setting [28, 29].

Interface automata and modal specifications are incomparable models as *must*, *may* and *input,output* have orthogonal meanings. Both models have advantages and disadvantages:

- Interface automata is a model that allows to make assumptions on the environment, which is mainly useful to derive a rich notion for composition. Unfortunately, the model is incomplete as conjunction, and quotient are not defined for this game-based model.

- Modal specification is a rich language algebra model on which most of requirements for a good interface theory can be considered. Unfortunately, *may* and *must* modalities are not sufficient to derive a rich notion for composition including compatibility.

It is thus worth considering unification of the frameworks of interface automata and modal specifications. A first attempt was made by Larsen et al. [23, 27] who considered *modal interfaces* that are modal specifications whose actions are also typed in *input* or *output* attributes. Larsen et al. have proposed a product-like construction allowing to address compatibility of modal interfaces. Nevertheless contrary to what is claimed by the authors, this composition operator in [23, 27] is not monotone with respect to the refinement of modal specifications. This fails to ensure that two *compatible* interfaces may be implemented separately.

The present paper adds a new stone to the cathedral of results on interface theories by (1) correcting the modal interface composition operator presented in [23, 27], (2) drawing a complete picture of the modal interface algebra , and (3) pushing even further the comparison between interface automata, modal automata and modal specifications and modal interfaces.

The rest of the paper is organized as follows. In Sections 2 and 3 we recap the theory for modal specifications and interface automata, respectively. In Section 4, we present the complete theory for modal interfaces and correct the error in [23, 27]. Finally, in Section 5, we draw our conclusion and discuss future extensions for the model of modal interfaces.

2. MODAL SPECIFICATIONS

This section overviews existing results for modal specifications. We start by introducing the framework, then we discuss the extension to several alphabets and study the notions of refinement and implementation. Finally, we present results on combining modal specifications.

2.1 The Framework

Following our previous work [30], we will define modal specifications in term of languages, knowing that they can also be interpreted as deterministic automata whose transitions are typed with *may* and *must* modalities. We propose the following definition.

DEFINITION 1. *A* modal specification *is a tuple* $\mathcal{S} = (A, must, may)$*, where* A *is a finite alphabet and:*

$$must, may : A^* \mapsto 2^A$$

are partial functions satisfying the following consistency *condition:*

$$must(u) \quad \subseteq \quad may(u). \tag{1}$$

The fact that $a \in may(u)$ means that action a is allowed after the trace u whereas $a \in must(u)$ indicates that a is required after u. By negation, $a \notin may(u)$ means that a is disallowed after u. The latter is often written $a \in mustnot(u)$. The condition (1) naturally imposes that every required action is also allowed. We shall sometimes write $A_{\mathcal{S}}, may_{\mathcal{S}}$, and $must_{\mathcal{S}}$ to refer to the entities involved in the definition of $\mathcal{S}$.

When composing specifications, discrepancies between the modal informations carried out by the specifications may appear. We then consider *pseudo-modal specifications*, denoted ${}^p\mathcal{S}$; they are triples satisfying Definition 1 with the exception of (1). For ${}^p\mathcal{S}$ a pseudo-modal specification, a word $u \in A^*$ is called *consistently specified* in ${}^p\mathcal{S}$ if it satisfies (1) and *inconsistent* otherwise; modal specifications correspond exactly to the subclass of *consistent* pseudo-modal specifications, that is pseudo-specifications such that every $u \in A^*$ is consistently specified.

A similar approach has been developed in [25] for a *non-modal* process algebraic framework in which a dedicated predicate is used to model inconsistent processes.

For ${}^p\mathcal{S} = (A, must, may)$ a pseudo-modal specification, the *support* of ${}^p\mathcal{S}$ is the least *prefix-closed* language $\mathcal{L}_{{}^p\mathcal{S}}$ such that: (i) $\epsilon \in \mathcal{L}_{{}^p\mathcal{S}}$, where ϵ denotes the empty word; and (ii) $u \in \mathcal{L}_{{}^p\mathcal{S}}$ and $a \in may(u)$ imply $u.a \in \mathcal{L}_{{}^p\mathcal{S}}$.

2.2 Multiple Alphabets

Large systems are composed of many subsystems possessing their own alphabets for ports and variables. The way those different alphabets are handled when combining subsystems requires some care.

We start with a series of definitions on languages. Let A and C be two alphabets such that $A \subseteq C$. For $v \in C^*$, the *projection* of v on A (denoted $\mathbf{pr}_A(v)$) is the word over A obtained from v by erasing all symbols that do not belong to A. Let $\mathcal{L}$ be a language over A, the *extension* of $\mathcal{L}$ to C is the language $\mathcal{L}_{\uparrow C} = \{v \in C^* \mid \mathbf{pr}_A(v) \in \mathcal{L}\}$.

DEFINITION 2. *The* shuffle product $\mathcal{L}_1 \times \mathcal{L}_2$ *of two languages* $\mathcal{L}_1 \subseteq A_1^*$ *and* $\mathcal{L}_2 \subseteq A_2^*$ *is given by*

$$\mathcal{L}_1 \times \mathcal{L}_2 = (\mathcal{L}_1)_{\uparrow A} \cap (\mathcal{L}_2)_{\uparrow A}, \text{ where } A = A_1 \cup A_2.$$

In modal automata, one has to consider two alphabet extensions: the weak and the strong extension. We shall see that the extension in use will depend on the operation that is performed on modal specification [30].

DEFINITION 3 (WEAK AND STRONG EXTENSIONS). *Let* ${}^p\mathcal{S} = (A, must_{{}^p\mathcal{S}}, may_{{}^p\mathcal{S}})$ *be a pseudo-modal specification and let* $C \supseteq A$.

1. *The* weak extension *of* ${}^p\mathcal{S}$ *to* C *is the pseudo-modal specification* ${}^p\mathcal{S}_{\Uparrow C} = (C, must, may)$ *such that* $\forall v \in C^*$*:*

$$\begin{cases} must(v) & = & must_{{}^p\mathcal{S}}(\mathbf{pr}_A(v)) \\ may(v) & = & may_{{}^p\mathcal{S}}(\mathbf{pr}_A(v)) \cup (C - A). \end{cases}$$

2. *The* strong extension *of* ${}^p\mathcal{S}$ *to* C *is the pseudo-modal specification* ${}^p\mathcal{S}_{\uparrow C} = (C, must, may)$ *such that* $\forall v \in C^*$*:*

$$\begin{cases} must(v) & = & must_{{}^p\mathcal{S}}(\mathbf{pr}_A(v)) \cup (C - A) \\ may(v) & = & may_{{}^p\mathcal{S}}(\mathbf{pr}_A(v)) \cup (C - A). \end{cases}$$

It is easy to show that $\mathcal{L}_{(\mathcal{S}_{\Uparrow C})} = \mathcal{L}_{(\mathcal{S}_{\uparrow C})} = (\mathcal{L}_{\mathcal{S}})_{\uparrow C}$.

2.3 Implementation and refinement

In this section, we study the concepts of *implementation*, *refinement* and *consistency*. We start with implementation, also called *model*.

DEFINITION 4 (IMPLEMENTATION). *Let* ${}^p\mathcal{S} = (A, must, may)$ *be a pseudo-modal specification.*

1. **Equal Alphabets:** *A prefix-closed language* $\mathcal{I} \subseteq A^*$ *is an* implementation *of* ${}^p\mathcal{S}$ *, denoted by* $\mathcal{I} \models {}^p\mathcal{S}$*, if* $\forall u \in \mathcal{I}, must(u) \subseteq \mathcal{I}_u \subseteq may(u)$*, where* $\mathcal{I}_u = \{a \in A \mid u.a \in \mathcal{I}\}$.

2. **Extended Alphabets:** *For* $C \supseteq A$*, a prefix-closed language* $\mathcal{I} \subseteq C^*$ *is a* weak implementation *of* ${}^p\mathcal{S}$*, written* $\mathcal{I} \models_w {}^p\mathcal{S}$*, iff* $\mathcal{I} \models {}^p\mathcal{S}_{\Uparrow C}$ *holds; it is a* strong implementation *of* ${}^p\mathcal{S}$*, written* $\mathcal{I} \models_s {}^p\mathcal{S}$*, iff* $\mathcal{I} \models {}^p\mathcal{S}_{\uparrow C}$ *holds.*

Modal specifications are equivalent to the fragment of the μ-calculus called the conjunctive ν-calculus [18]. Hence, a model for a modal specification is a model for the formula represented by the specification.

Satisfaction can be related to consistently specified words:

LEMMA 1. *If* $\mathcal{I} \models {}^p\mathcal{S}$*, then* $\mathcal{I} \subseteq \mathcal{L}_{{}^p\mathcal{S}}$ *holds and every word of* $\mathcal{I}$ *is consistently specified in* ${}^p\mathcal{S}$*. Similarly, if* $\mathcal{I} \models_w {}^p\mathcal{S}$ *or* $\mathcal{I} \models_s {}^p\mathcal{S}$*, then* $\mathcal{I} \subseteq (\mathcal{L}_{{}^p\mathcal{S}})_{\uparrow C}$ *holds and for every word* $v \in C^*$ *of* $\mathcal{I}$*,* $\mathbf{pr}_A(v)$ *is consistently specified in* ${}^p\mathcal{S}$.

We now switch to the case of modal refinement which extends in a natural manner the classical notion of bisimulation on automata. We first consider the case where specifications are defined over the same alphabet:

DEFINITION 5. *Let* ${}^p\mathcal{S}_1 = (A, must_1, may_1)$ *and* ${}^p\mathcal{S}_2 = (A, must_2, may_2)$ *be two pseudo-modal specifications then* ${}^p\mathcal{S}_1$ refines ${}^p\mathcal{S}_2$*, denoted* ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_2$*, iff for all* $u \in \mathcal{L}_{{}^p\mathcal{S}_1}$*:*

$$\begin{array}{rcl} may_1(u) & \subseteq & may_2(u) \\ must_1(u) & \supseteq & must_2(u). \end{array}$$

It can be shown that refinement is a preorder relation which implies the inclusion of supports. As a consequence, any two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \leq \mathcal{S}_2 \leq \mathcal{S}_1$ have equal supports $\mathcal{L} = \mathcal{L}_{\mathcal{S}_1} = \mathcal{L}_{\mathcal{S}_2}$ and moreover, for

all $u \in \mathcal{L}$, $may_1(u) = may_2(u)$ and $must_1(u) = must_2(u)$. Thus equivalent modal specifications differ only outside of their support; a unique representant $\mathcal{S} = (A, must, may)$ of equivalence classes of modal specifications can be defined by assuming that for all $u \notin \mathcal{L}_\mathcal{S}$, $must(u) = \emptyset$ and $may(u) = A$. Under this assumption, modal refinement is a *partial order relation* on modal specifications. In the following, only modal specifications in this *canonical* form are considered.

DEFINITION 6. *Let ${}^p\mathcal{S}_1 = (A_1, must_1, may_1)$ and ${}^p\mathcal{S}_2 = (A_2, must_2, may_2)$ be two pseudo-modal specifications with $A_1 \supseteq A_2$ then ${}^p\mathcal{S}_1$* weakly refines *${}^p\mathcal{S}_2$ (which is denoted ${}^p\mathcal{S}_1 \leq_w {}^p\mathcal{S}_2$), iff ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_{2\Uparrow A_1}$, and it* strongly refines *${}^p\mathcal{S}_2$, written ${}^p\mathcal{S}_1 \leq_s {}^p\mathcal{S}_2$, iff ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_{2\uparrow A_1}$.*

A pseudo-modal specification can be reduced into a modal specification with preservation of its semantic:

THEOREM 2 (CONSISTENCY). *Either a pseudo-modal specification ${}^p\mathcal{S}$ has no model, or there exists a modal specification $\rho({}^p\mathcal{S})$ having the same alphabet of actions such that $\rho({}^p\mathcal{S})$ possesses the same set of weak and strong implementations:*

$$\begin{array}{rcl} \mathcal{I} \models_w {}^p\mathcal{S} & \Leftrightarrow & \mathcal{I} \models_w \rho({}^p\mathcal{S}) \\ \mathcal{I} \models_s {}^p\mathcal{S} & \Leftrightarrow & \mathcal{I} \models_s \rho({}^p\mathcal{S}) \end{array}$$

We shall call $\rho({}^p\mathcal{S})$ the reduction of ${}^p\mathcal{S}$. The detailed construction of $\rho({}^p\mathcal{S})$ can be found in [30]. We let $\perp$ be a particular modal specification that admits no model and let $\mathcal{L}_\perp$ be the empty set.

We conclude the section with the following theorem that relates refinement and implementation.

THEOREM 3 (IMPLEMENTATION AND REFINEMENT).

1. *Weak and strong implementation and refinement are related as follows: $\models_s \subseteq \models_w$ and $\leq_s \subseteq \leq_w$.*
2. *Weak and strong modal refinement are both sound and complete w.r.t. weak and strong thorough refinement, respectively:*

$$\begin{array}{rcl} \mathcal{S}_2 \leq_w \mathcal{S}_1 & \Leftrightarrow & \{\mathcal{I} \mid \mathcal{I} \models_w \mathcal{S}_2\} \subseteq \{\mathcal{I} \mid \mathcal{I} \models_w \mathcal{S}_1\} \\ \mathcal{S}_2 \leq_s \mathcal{S}_1 & \Leftrightarrow & \{\mathcal{I} \mid \mathcal{I} \models_s \mathcal{S}_2\} \subseteq \{\mathcal{I} \mid \mathcal{I} \models_s \mathcal{S}_1\}. \end{array}$$

As already noticed, modal specifications are equivalent to deterministic modal automata. When allowing for nondeterminism, the theorem above does not hold as modal refinement is no more complete [24].

2.4 Operations on modal specifications

Consider two modal specifications $\mathcal{S}_1 = (A_1, must_1, may_1)$ and $\mathcal{S}_2 = (A_2, must_2, may_2)$, we now define their *conjunction*, *parallel product* and *quotient*. We proceed in two steps: we first define these operations when $A_1 = A_2$; the case of different alphabets is then handled by performing a preliminary step of alphabet equalization.

In [30], we argued that alphabet equalization must be different depending on the considered operation. Such an extension must be *neutral*, meaning that it should not constrain what other interfaces may want to require regarding these extra actions.

Conjunction.

When $A_1 = A_2$, the *conjunction* $\mathcal{S}_1 \wedge \mathcal{S}_2 = \rho(\mathcal{S}_1 \& \mathcal{S}_2)$ where $\mathcal{S}_1 \& \mathcal{S}_2$ is defined by:

$$\begin{array}{rcl} may_{\mathcal{S}_1 \& \mathcal{S}_2}(u) & = & may_1(u) \cap may_2(u) \\ must_{\mathcal{S}_1 \& \mathcal{S}_2}(u) & = & must_1(u) \cup must_2(u). \end{array} \quad (2)$$

Observe that it is not guaranteed that $\mathcal{S}_1 \& \mathcal{S}_2$ satisfies (1). Hence, we use theorem 2 and apply the reduction operation ρ in order to obtain a modal specification.

For the general case where $A_1 \neq A_2$, the definition above is applied after an equalization step: $\mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{S}_{1\Uparrow A} \wedge \mathcal{S}_{2\Uparrow A}$, with $A = A_1 \cup A_2$.

THEOREM 4.

$$\mathcal{I} \models_w \mathcal{S}_1 \wedge \mathcal{S}_2 \quad \Leftrightarrow \quad \mathcal{I} \models_w \mathcal{S}_1 \text{ and } \mathcal{I} \models_w \mathcal{S}_2.$$

The conjunction between $\mathcal{S}_1$ and $\mathcal{S}_2$ is exactly their *greatest lower bound* for the weak refinement relation: $\mathcal{S}_1 \wedge \mathcal{S}_2$ is the greatest specification that weakly refines both $\mathcal{S}_1$ and $\mathcal{S}_2$.

A current practice in the design of a component is to give several specifications, each of them describing a particular requirement. The conjunction of these specifications, enables to check the consistency of these requirements, by deciding satisfiability.

Parallel product.

When $A_1 = A_2$, the *parallel product* $\mathcal{S} = \mathcal{S}_1 \otimes \mathcal{S}_2$ is defined by:

$$\begin{array}{rcl} may_{\mathcal{S}}(u) & = & may_1(u) \cap may_2(u) \\ must_{\mathcal{S}}(u) & = & must_1(u) \cap must_2(u). \end{array} \quad (3)$$

The product of two modal specifications always satisfy the consistency condition. Hence, no reduction is needed. For the general case where $A_1 \neq A_2$, the definition above is applied after an equalization step: $\mathcal{S}_1 \otimes \mathcal{S}_2 = \mathcal{S}_{1\uparrow A} \otimes \mathcal{S}_{2\uparrow A}$.

In an interface theory, it is desirable to be able to develop components in isolation and then to compose them as expected. This is ensured by the product operation as stated with the following theorem.

THEOREM 5.

1. *If $\mathcal{S}'_1 \leq_s \mathcal{S}_1$ and $\mathcal{S}'_2 \leq_s \mathcal{S}_2$, then $\mathcal{S}'_1 \otimes \mathcal{S}'_2 \leq_s \mathcal{S}_1 \otimes \mathcal{S}_2$.*
2. *If $\mathcal{I}_1 \models_s \mathcal{S}_1$ and $\mathcal{I}_2 \models_s \mathcal{S}_2$, then $\mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S}_1 \otimes \mathcal{S}_2$.*
3. *Regarding supports: $\mathcal{L}_{\mathcal{S}_1 \otimes \mathcal{S}_2} = \mathcal{L}_{\mathcal{S}_1} \times \mathcal{L}_{\mathcal{S}_2}$.*

Strong refinement has to be used when enlarging the alphabet, as the product is not monotonic with respect to the weak refinement [30].

Residuation/quotient.

The operation of *residuation*, also called *quotient*, is the adjoint of product. Intuitively, the quotient enables to describe a part of a global specification assuming another part is already realized by some component. If $A_1 = A_2$, then

the *pseudo*-quotient ${}^{p}\mathcal{S} = \mathcal{S}_1 /\!\!/ \mathcal{S}_2$ is defined by:

$$\begin{array}{lll}
a \in may_{{}^{p}\mathcal{S}}(u) \cap must_{{}^{p}\mathcal{S}}(u) & \text{if} & a \in must_1(u) \\
 & \text{and} & a \in must_2(u) \\
a \in must_{{}^{p}\mathcal{S}}(u) \setminus may_{{}^{p}\mathcal{S}}(u) & \text{if} & a \in must_1(u) \\
 & \text{and} & a \notin must_2(u) \\
a \in may_{{}^{p}\mathcal{S}}(u) \setminus must_{{}^{p}\mathcal{S}}(u) & \text{if} & a \in may_1(u) \\
 & \text{and} & a \notin must_1(u) \\
a \in may_{{}^{p}\mathcal{S}}(u) \setminus must_{{}^{p}\mathcal{S}}(u) & \text{if} & a \notin may_1(u) \\
 & \text{and} & a \notin may_2(u) \\
a \notin may_{{}^{p}\mathcal{S}}(u) \cup must_{{}^{p}\mathcal{S}}(u) & \text{if} & a \notin may_1(u) \\
 & \text{and} & a \in may_2(u).
\end{array}$$

Due to the second rule, $\mathcal{S}_1 /\!\!/ \mathcal{S}_2$ may have inconsistently specified words. As a consequence, a reduction operation may be needed and the quotient of $\mathcal{S}_1$ by $\mathcal{S}_2$ is $\mathcal{S}_1/\mathcal{S}_2 = \rho(\mathcal{S}_1 /\!\!/ \mathcal{S}_2)$.
For the general case of two different alphabets, the definition above is applied after an alphabet equalization step: $\mathcal{S}_1 \,/\, \mathcal{S}_2 = \mathcal{S}_{1\Uparrow A} \,/\, \mathcal{S}_{2\uparrow A}$.
We have the following theorems:

THEOREM 6. *Let $\mathcal{S}$, $\mathcal{S}_1$ and $\mathcal{S}_2$ be modal specifications such that $A_{\mathcal{S}_2} \supseteq A_{\mathcal{S}} \supseteq A_{\mathcal{S}_1}$. We have*

$$\mathcal{S}_2 \leq_s \mathcal{S}/\mathcal{S}_1 \Leftrightarrow \mathcal{S}_1 \otimes \mathcal{S}_2 \leq_s \mathcal{S}.$$

THEOREM 7. *Let $\mathcal{S}$, $\mathcal{S}_1$ be modal specifications and $\mathcal{I}_2$ a prefix-closed language such that $A_{\mathcal{I}_2} \supseteq A_{\mathcal{S}} \supseteq A_{\mathcal{S}_1}$, we have*

$$\mathcal{I}_2 \models_s \mathcal{S}/\mathcal{S}_1 \Leftrightarrow [\forall \mathcal{I}_1 : \mathcal{I}_1 \models_s \mathcal{S}_1 \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S}].$$

3. INTERFACE AUTOMATA

In [14], de Alfaro and Henzinger introduced *interface automata*, that are automata whose transitions are typed with *input* and *output* actions rather than with modalities. In this section, we briefly overview the theory of interface automata and refer the reader to [14, 11] for more details.

DEFINITION 7. *An* interface automaton *is a tuple $\mathcal{P} = (X, x_0, A, \rightarrow)$, where X is the set of* states*, $x_0 \in X$ is the* initial state*, A is the alphabet of* actions*, and $\rightarrow \subseteq X \times A \times X$ is the transition relation.*

We decompose $A = A? \uplus A!$, where $A?$ is the set of inputs and $A!$ is the set of outputs. In the rest of the paper, we shall often use $a?$ to emphasize that $a \in A?$ and $a!$ for $a \in A!$. Observe that if we consider deterministic interface automata, then we can propose a language-based definition similar to the one we gave for modal specifications.

The semantic of an interface automaton is given by a two-player game between: an *input* player that represents the environment (the moves are the input actions), and an *output* player that represents the component itself (the moves are the output actions). Input and output moves are in essence orthogonal to modalities. Interface automata are operational models, they do not encompass any notion of model, and thus neither satisfiability nor consistency, because one cannot distinguish between interfaces and components implementations. Alternatively, properties of interfaces are described in game-based logics, *e.g.*, ATL [1], with a high-cost complexity. Refinement between interface automata corresponds to the alternating refinement relation between games [2], *i.e.*, an interface refines another one if its environment is more permissive whereas its component is more restrictive. There is no notion of component reuse and shared refinement is defined in an ad-hoc manner [16].

The main advantage of the game-based approach appears in the definition of composition and *compatibility* between components.

Following [11], two interface automata are composable if they have disjoint sets of output actions compose by synchronizing on shared actions and interleave asynchronously all other actions.

DEFINITION 8 (PRODUCT OF INTERFACE AUTOMATA). *Let $\mathcal{P}_1 = (X_1, x_{01}, A_1, \rightarrow_1)$ and $\mathcal{P}_2 = (X_2, x_{02}, A_2, \rightarrow_2)$ be two interface automata. The product between $\mathcal{P}_1$ and $\mathcal{P}_2$ is an interface automaton $\mathcal{P}_1 \times \mathcal{P}_2 = (X, x_0, A, \rightarrow)$, where*

- *$X = X_0 \times X_1$;*
- *$x_0 = x_{01} \times x_{02}$;*
- *$A = A_1 \cup A_2$, and $A? = (A_1? \cup A_2?) \setminus ((A_1? \cap A_2!) \cup (A_2? \cap A_1!))$, and $A! = A_1! \cup A_2!$;*
- *$\rightarrow$ is defined as follows:*
  - *For each action $a \in A$ such that $a \notin A_1 \cap A_2$, there exists a transition $(x_1, y_1) \xrightarrow{a} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{a}_1 (x_2)$ and $y_1 = y_2$ or $(y_1) \xrightarrow{a}_2 (y_2)$ and $x_1 = x_2$.*
  - *For each action $a \in A_1? \cap A_2?$, there exists a transition $(x_1, y_1) \xrightarrow{a?} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{a?}_1 (x_2)$ and $(y_1) \xrightarrow{a?}_2 (y_2)$.*
  - *For each $a \in (A_1? \cap A_2!) \cup (A_2? \cap A_1!)$, there exists a transition $(x_1, y_1) \xrightarrow{a!} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{a}_1 (x_2)$ and $(y_1) \xrightarrow{a}_2 (y_2)$.*

Since interface automata are not necessarily input-enabled[1] (which allows to make assumptions on the environment), in the product $\mathcal{P}_1 \times \mathcal{P}_2$ of two interface automata $\mathcal{P}_1$ and $\mathcal{P}_2$, there may be *illegal states* where one of the automata may produce an output action that is also in the input alphabet of the other automaton, but is not accepted at this state. In most of existing models for interface theories that are based on an input output setting, the interfaces would be declared to be *incompatible*. This is a pessimistic approach that can be avoided by exploiting the game-based semantic. Indeed, the game semantic allows to propose an optimistic approach:

> "Two interfaces can be composed and are compatible if there is at least one environment where they can work together (i.e., where they can avoid the illegal states)."

Deciding whether there exists an environment where the two interfaces can work together is equivalent to checking whether the environment in the product of the interfaces has a strategy to always avoid illegal states. The set of states from which the environment has a strategy to avoid the illegal states whatever the component does can be recursively computed as follows.

[1]Recall that a system is input-enabled if it can react to any input action in any moment.

Let $\mathit{Illegal}(\mathcal{P}_1, \mathcal{P}_2)$ is the subset of pairs $(x_1, x_2) \in X_1 \times X_2$ such that there exists

$$\begin{array}{rlll} \text{either an action } a \in A_1! \cap A_2? & \text{with} & x_1 \xrightarrow{a!}_1 \\ & \text{but not} & x_2 \xrightarrow{a?}_2 \\ \text{or an action } a \in A_2! \cap A_1? & \text{with} & x_2 \xrightarrow{a!}_2 \\ & \text{but not} & x_1 \xrightarrow{a?}_1 \end{array}$$

where $x \xrightarrow{a}$ means that $x \xrightarrow{a} y$ for some state y. If illegal states exist in the product $\mathcal{P}_1 \times \mathcal{P}_2$, there may still exist refinements of it that possess no illegal states. Such a refinement specifies how the use of the resulting product should be restricted in order to guarantee that illegal states cannot be reached. As proved in [14], such a largest refinement is obtained by backward pruning $\mathcal{P}_1 \times \mathcal{P}_2$ as follows. For $Y \subseteq X$, the set of states of $\mathcal{P}_1 \times \mathcal{P}_2$, let $pre_!(Y)$ be the subset $Z \subseteq X$ of states z such that $z \xrightarrow{a!} y$ for some $y \in Y$ and $a! \in A!$ (an output action of the product). Let $pre_!^0(Y) = Y$ and, for $k \geq 0$, $pre_!^{k+1}(Y) = pre_!\big(pre_!^k(Y)\big)$ and let $pre_!^*(Y) = \bigcup_k pre_!^k(Y)$.

The desired pruning consists in:

- Removing $pre_!^*(\mathit{Illegal}(\mathcal{P}_1, \mathcal{P}_2))$ from X, and
- Removing transitions to states in $pre_!^*(\mathit{Illegal}(\mathcal{P}_1, \mathcal{P}_2))$, and
- Removing unreachable states.

The result of applying the pruning to $\mathcal{P}_1 \times \mathcal{P}_2$ is denoted by

$$\mathcal{P}_1 \parallel \mathcal{P}_2,$$

and is called the *composition* of the two interface automata. $\mathcal{P}_1$ and $\mathcal{P}_2$ are called *compatible* if applying the pruning leaves the initial state [14].

We recall the two following theorems from [14] that show that interface automata support independent design and substituability.

THEOREM 8 ([14]). *The composition operation is associative and commutative.*

THEOREM 9 ([14]). *Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ be three interface automata. If $\mathcal{P}_2$ refines $\mathcal{P}_1$ and the set of shared actions $\mathcal{P}_2 \parallel \mathcal{P}3$ of is included in the set of shared actions of $\mathcal{P}_1 \parallel \mathcal{P}_3$, then $\mathcal{P}_2 \parallel \mathcal{P}_3$ refines $\mathcal{P}_1 \parallel \mathcal{P}_3$.*

REMARK 1. *The operations between interface automata that have been defined so far do not require an explicit treatment of dissimilar alphabets as it is the case for modal specifications.*

4. ON MODAL INTERFACES

We now present the full theory for *modal interfaces*. Modal interfaces is an extension of modal specifications where actions are also typed with *input* and *output*. This addition allows to propose notions of composition and compatibility for modal specifications in the spirit of interface automata.

The first account on compatibility for modal interfaces was proposed in [23, 27]. In this section, we propose a full interface theory for modal interfaces, which includes composition, product, conjunction, and component reuse via quotient. Moreover, we show that the composition operator proposed in [23, 27] is incorrect and we propose a correction.

We shall start our theory with the definition of *profiles* which are used to type actions of modal specifications with *input* and *output*:

4.1 Profiles

For an alphabet of actions A, a *profile* is a function $\pi : A \mapsto \{?, !\}$, labeling actions with the symbols ? (for *inputs*) or ! (for *outputs*). We write "$a?$" to express that "$\pi(a) =?$", and similarly for the other case. We denote by $A?$ the set of $a \in A$ such that $\pi(a) =?$ and similarly for $A!$. We shall sometimes write by abuse of language, $\pi = (A?, A!)$.

We now discuss operations on profiles. We consider a profile $\pi_1 = (A_1?, A_1!)$ defined over A_1 and a profile $\pi_2 = (A_2?, A_2!)$ defined over A_2.

Product between profiles. The composition between π_1 and π_2, which is defined iff $A_1! \cap A_2! = \emptyset$, is the $\pi = (A?, A!)$ such that

$$\pi_1 \otimes \pi_2 : \left\{ \begin{array}{rcl} A! & = & (A_1! \ \cup \ A_2!) \\ A? & = & (A_1? \ \cup \ A_2?) \setminus A! \end{array} \right.$$

Refinement between profiles. Profile π_2 refines π_1 (denoted $\pi_2 \leq \pi_1$) iff $A_2 \supseteq A_1$ and both profiles coincide on A_1: $\forall a \in A_1, \pi_2(a) = \pi_1(a)$.

Conjunction between profiles. The conjunction between π_1 and π_2 (denoted $\pi_1 \wedge \pi_2$) is the greatest lower bound of the profiles, whenever it exists. More precisely, the conjunction of profiles π_1 and π_2 is defined iff both profiles coincide on their common alphabet: $\forall a \in A_1 \cap A_2, \pi_1(a) = \pi_2(a)$. Whenever defined, the conjunction $\pi_1 \wedge \pi_2$ coincides with π_1 for every letter in A_1 and with π_2 for every letter in A_2.

Quotient between profiles. The *quotient* $\pi_1 \,/\, \pi_2$ is defined as the adjoint of $\otimes$, if it exists, namely $\pi_1 \,/\, \pi_2 = \max\{\pi \mid \pi \otimes \pi_2 \leq \pi_1\}$. More precisely, $\pi_1 \,/\, \pi_2$ is defined if and only if $A_1 \supseteq A_2$ and $A_1! \supseteq A_2!$, and is then equal to the profile $\pi = (A?, A!)$ such that

$$\pi_1 \,/\, \pi_2 : \left\{ \begin{array}{rcl} A! & = & A_1! \setminus A_2! \\ A? & = & A_1? \setminus A_2?. \end{array} \right.$$

4.2 The framework of modal interfaces

We now formally introduce modal interfaces that are modal specification whose actions are also labeled with *input* and *output* attributes. We will consider the language representation in the spirit of [28, 30], while Larsen et al. followed the automata-based representation (the two representations are equivalent).

DEFINITION 9 (MODAL INTERFACE). *A* modal interface *is a pair $\mathcal{C} = (\mathcal{S}, \pi)$, where $\mathcal{S}$ is a modal specification on the alphabet $A_{\mathcal{S}}$ and $\pi : A_{\mathcal{S}} \to \{?, !\}$ is a* profile.

A model for a modal interface is a tuple $(\mathcal{I}, \pi')$, where $\mathcal{I}$ is a prefix-closed language and π' is a profile for $\mathcal{I}$. We say that $(\mathcal{I}, \pi')$ *strongly implements* $(\mathcal{S}, \pi)$, written $(\mathcal{I}, \pi') \models_s (\mathcal{S}, \pi)$, if $\mathcal{I} \models_s \mathcal{S}$ and $\pi' \leq \pi$, and similarly for *weak implementation*. We say that $(\mathcal{S}_2, \pi_2) \leq_s (\mathcal{S}_1, \pi_1)$ if $\mathcal{S}_2 \leq_s \mathcal{S}_1$ and $\pi_2 \leq \pi_1$, with corresponding definition for weak refinement $\leq_w$. The *composition* of two models is the pair that results from the shuffle product $\times$ of their prefix-closed languages and of the product of their profiles.

4.3 Operations on modal interfaces

Operations on modal specifications directly extend to operations on modal interfaces. We have the following definition.

DEFINITION 10. *Consider two modal interfaces* $\mathcal{C}_1 = (\mathcal{S}_1, \pi_1)$ *and* $\mathcal{C}_2 = (\mathcal{S}_2, \pi_2)$*, and let* $\star \in \{\wedge, \otimes, /\}$*. If* $\pi_1 \star \pi_2$ *is defined, then*

$$\mathcal{C}_1 \star \mathcal{C}_2 = (\mathcal{S}_1 \star \mathcal{S}_2, \pi_1 \star \pi_2).$$

All the nice properties of modal specifications directly extend to modal interfaces.

THEOREM 10. *Theorems 1 to 6 extend to modal interfaces.*

4.4 On compatibility for modal interfaces

In this section, we take advantage of profiles to define a notion of composition with compatibility issue for modal interfaces. We shall recap the solution proposed in [23, 27], then we shall show a counter example to Theorem 10 in [23] and then propose our correction. We first recap the translation from interface automata to modal interfaces, which will help to make the link between modalities and input or output actions.

4.4.1 From interface automata to modal interfaces

We recap the translation from interface automata to modal automata that has been proposed in [23]. In this section, we extend this translation to modal specification, the language-extension corresponding to modal automata.

We consider an interface automaton $\mathcal{P} = (X, x_0, A, \rightarrow)$. We assume $\mathcal{P}$ to be deterministic and we let $\mathcal{L}_\mathcal{P}$ denote the (prefix-closed) language defined by $\mathcal{P}$. The alphabet of $\mathcal{S}_\mathcal{P}$ is $A_{\mathcal{S}_\mathcal{P}} = A$ and modalities are defined for all $u \in A^*_\mathcal{P}$:

$$\begin{array}{lll} a? \in must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a? \in \mathcal{L}_\mathcal{P} \\ a! \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a! \in \mathcal{L}_\mathcal{P} \\ a? \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\ & \text{and} & u.a? \notin \mathcal{L}_\mathcal{P} \\ a! \notin may_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\ & \text{and} & u.a! \notin \mathcal{L}_\mathcal{P} \\ a \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \notin \mathcal{L}_\mathcal{P}. \end{array} \quad (4)$$

Theorem 1 of [23] shows that, with the above correspondence, alternating simulation for interface automata and modal refinement for modal interfaces coincide. Regarding supports, we have:

$$\mathcal{L}_{\mathcal{S}_\mathcal{P}} = \mathcal{L}_\mathcal{P} \uplus \{u.a?.v \mid u \in \mathcal{L}_\mathcal{P}, u.a? \notin \mathcal{L}_\mathcal{P}, v \in A^*_\mathcal{P}\}. \quad (5)$$

It is worth making some comments about this translation, given by formulas (4,5). Regarding formula (5), the supporting language $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ allows the environment to violate the constraints set on it by the interface automaton $\mathcal{P}$. When this happens—formally, the environment exits the alternating simulation relation—the component considers that the assumptions under which it was supposed to perform are violated, so it allows itself breaching its own promises and can perform anything afterward. One could also see the violation of assumptions as an exception. Then, $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ states no particular exception handling since everything is possible. Specifying exception handling then amounts to refining this modal interface.

Formula (4) refines (5) by specifying obligations. Case 1 expresses that the component *must* accept from the environment any input within the assumptions. Case 2 indicates that the component behaves according to best effort regarding its own outputs actions. Finally, cases 3 and 4 express that the violation of its obligations by the environment are seen as an exception, and that exception handling is unspecified and not mandatory.

4.4.2 The composition by Larsen et al. and the bug in Theorem 10 of [23]

We now consider the notion of compatibility for two Modal Interfaces $\mathcal{C}_1 = (\mathcal{S}_1, \pi_1)$ and $\mathcal{C}_2 = (\mathcal{S}_2, \pi_2)$ with $\mathcal{S}_1$ defined over A_1 and $\mathcal{S}_2$ defined over A_2. We assume that $\mathcal{C}_1$ and $\mathcal{C}_2$ do not share common output actions (which is the composability requirement similar to the one for interface automata). We first compute the product between $\mathcal{C}_1$ and $\mathcal{C}_2$ following Definition 4.3.

We then define $Illegal(\mathcal{C}_1, \mathcal{C}_2)$ to be the subset of words u belonging to the support of $\mathcal{C}_1 \otimes \mathcal{C}_2$, such that there exists

$$\begin{array}{rl} \text{either} & \text{an action } a \in A_1! \cap A_2? \\ & \text{with } a \in may_1(u_1) \setminus must_2(u_2) \\ \text{or} & \text{an action } a \in A_2! \cap A_1? \\ & \text{with } a \in may_2(u_2) \setminus must_1(u_1), \end{array} \quad (6)$$

where $u_1 = \mathbf{pr}_{A_1}(u)$ and similarly $u_2 = \mathbf{pr}_{A_2}(u)$. Getting rid of illegal runs is performed as follows. For U a set of words of Modal Interface $\mathcal{C}$, let $pre_!(U)$ be the set

$$pre_!(U) \quad = \quad \{v \in \mathcal{L}_\mathcal{C} \mid \exists a! \in may(v), v.a! \in U\}$$

Let $pre^0_!(U) = U$, and, for $k \geq 0$, $pre^{k+1}_!(U) = pre_!\big(pre^k_!(U)\big)$. Finally, let $pre^*_!(U) = \bigcup_k pre^k_!(U)$.

The composition of two modal interfaces is obtained from their product by removing states in $pre^*_!(U)$, following the approach outlined for interface automata. Two modal interfaces are compatible if the pruning with the illegal words do not remove the empty word. The composition between $\mathcal{C}_1$ and $\mathcal{C}_2$ is denoted $\mathcal{C}_1 \parallel \mathcal{C}_2$.

Theorem 10 in [23, 27] says that

> "*(Independent Implementability). For any two composable modal interfaces* $\mathcal{C}_1$*,* $\mathcal{C}_2$ *and two implementations* $(\mathcal{I}_1, \pi_1)$ *and* $(\mathcal{I}_2, \pi_2)$*. If* $(\mathcal{I}_1, \pi_1) \leq \mathcal{C}_1$ *and* $(\mathcal{I}_2, \pi_2) \leq \mathcal{C}_2$*, then it holds that* $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \leq \mathcal{C}_1 \parallel \mathcal{C}_2$."

The following example shows that Theorem 10 in [23, 27] is wrong.

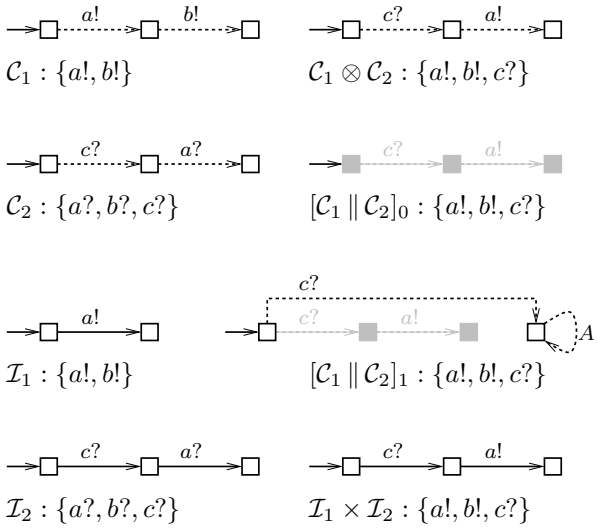

Figure 1: Counterexample regarding compatibility. Grey-shaded states are to be removed.

EXAMPLE 1. *Figure 1 depicts two Modal Interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$; may \ must actions are depicted using dashed arrows whereas solid arrows corresponds to must actions. $\mathcal{I}_1$ and $\mathcal{I}_2$ are implementations of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Alphabets are indicated for each modal interface. Parallel composition according to [23] is named $[\mathcal{C}_1 \parallel \mathcal{C}_2]_0$. Word $c?.a!$ is illegal since in the state reached after this run $\mathcal{C}_1$ may offer $b!$ whereas $\mathcal{C}_2$ may (in fact will) not accept it. However, $c?.a!$ is in the product of the two implementations.*

4.4.3 The correction

Call *exception* any word in $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ from which the environment has no strategy to prevent the occurrence of an illegal word, meaning that an illegal word can be obtained from the exception by following only output actions.

DEFINITION 11 (COMPATIBILITY). *The* exception language *of modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$ is the language $\mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2} = pre_!^*(Illegal(\mathcal{C}_1, \mathcal{C}_2))$. Modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$ are said to be* compatible *if and only if the empty word ϵ is not in $\mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$.*

DEFINITION 12 (PARALLEL COMPOSITION). *Given two modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$, the* relaxation *of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is obtained by applying the following pseudo-algorithm to $\mathcal{C}_1 \otimes \mathcal{C}_2$:*

```
for all v in L_{C1⊗C2} do
    for all a in A do
        if v ∉ E_{C1 ∥ C2} and v.a ∈ E_{C1 ∥ C2} then
            for all w in A* do
                must(v.a.w) := ∅
                may(v.a.w) := A
            end for
        end if
    end for
end for
```

If $\mathcal{C}_1$ and $\mathcal{C}_2$ are compatible, the relaxation of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is called the parallel composition *of $\mathcal{C}_1$ and $\mathcal{C}_2$, denoted by $\mathcal{C}_1 \parallel \mathcal{C}_2$. Whenever $\mathcal{C}_1$ and $\mathcal{C}_2$ are incompatible, the parallel composition $\mathcal{C}_1 \parallel \mathcal{C}_2$ is defined as the inconsistent modal specification $\perp$.*

If the environment performs an $a?$ to which the "**if** ... **then** ..." statement applies, then illegal words may exist for certain pairs $(\mathcal{I}_1, \mathcal{I}_2)$ of strong implementations of $\mathcal{C}_1$ and $\mathcal{C}_2$. If this occurs, then $\mathcal{C}_1 \parallel \mathcal{C}_2$ relaxes all constraints on the future of the corresponding runs — Nothing is forbidden, nothing is mandatory: the system has reached a "universal" state. This parallels the pruning rule combined with alternating simulation, in the context of interface automata.

EXAMPLE 2. *We now show that our relaxation allows to correct the counter example stated in Figure 1. We observe that our relaxation procedure yields $[\mathcal{C}_1 \parallel \mathcal{C}_2]_1$, with $A = \{a!, b!, c?\}$, which has $\mathcal{I}_1 \times \mathcal{I}_2$ as an implementation.*

Associativity of the parallel composition operator is one of the key requirements of an interface framework, since it enables independent design of sub-systems. Unlike in [23, 27], where associativity is only mentioned, we can now state the following theorem:

THEOREM 11. *The parallel composition operator is commutative and associative.*

Thanks to the interplay between modalities and profiles, knowledge about exceptions is preserved by parallel composition. This is the very reason why it is associative. Indeed, the last $a?$ action in exception runs of the from $v.a?$ comes with a may modality. In this way, it is distinguished from normal inputs which come with a must modality. When taking the parallel composition with another modal interface with a profile such that $a?$ is also an input, the resulting modality is a may. In this way, knowledge of the occurrence of an exception is preserved. Whenever this input action $a?$ is composed with an output $a!$, this results in an illegal run, meaning that an exception will be triggered earlier.

As for interface automata (Theorem 4 in [14]), strong refinement preserves compatibility, assuming that the refined modal interface does not introduce new shared actions.

LEMMA 12. *Given any three modal interfaces $\mathcal{C}_i$, $i = 1...3$, such that $\mathcal{C}_2 \leq_s \mathcal{C}_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$:*

- $\mathbf{pr}_{A_1 \cup A_3}(Illegal(\mathcal{C}_2, \mathcal{C}_3))$ *is included in* $Illegal(\mathcal{C}_1, \mathcal{C}_3)$*;*
- $\mathbf{pr}_{A_1 \cup A_3}(\mathcal{E}_{\mathcal{C}_2 \parallel \mathcal{C}_3})$ *is included in* $\mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_3}$*.*

Proof: *Consider an illegal word $u \in Illegal(\mathcal{C}_2, \mathcal{C}_3)$ for $\mathcal{C}_2 \otimes \mathcal{C}_3$. This means that there exists an action $a \in A_2 \cap A_3$ such that (i) either a is an output of $\mathcal{C}_2$ and an input of $\mathcal{C}_3$, such that $a \in may_2(\mathbf{pr}_{A_2}(u))$ and $a \notin must_3(\mathbf{pr}_{A_3}(u))$, or (ii) a is an input of $\mathcal{C}_2$ and an output of $\mathcal{C}_3$, such that $a \notin must_2(\mathbf{pr}_{A_2}(u))$ and $a \in may_3(\mathbf{pr}_{A_3}(u))$.*

By Definition 6, u is also in $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_{3\uparrow A2 \cup A3}}$. By Definition 3, $u' = \mathbf{pr}_{A_1 \cup A_3}(u)$ belongs to $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_3}$.

Since it is assumed that $A_2 \cap A_3 \subseteq A_1 \cap A_3$, action a belongs to $A_1 \cap A_3$. By Definition 5, either a is an output of $\mathcal{C}_1$ and an input of $\mathcal{C}_3$, such that $a \in may_1(\mathbf{pr}_{A_1}(u'))$ and $a \notin must_3(\mathbf{pr}_{A_3}(u'))$, or (ii) a is an input of $\mathcal{C}_1$ and an output of $\mathcal{C}_3$, such that $a \notin must_1(\mathbf{pr}_{A_1}(u'))$ and $a \in may_3(\mathbf{pr}_{A_3}(u'))$. Meaning that $u' \in Illegal(\mathcal{C}_1, \mathcal{C}_3)$, which proves the first part of the Lemma.

Next, recall that $A_1! \cup A_3!$ is included in $A_2! \cup A_3!$. Hence, the set $\mathbf{pr}_{A_1 \cup A_3}(pre_!^*(Illegal(\mathcal{C}_2, \mathcal{C}_3)))$ is included in the set $pre_!^*\big(\mathbf{pr}_{A_1 \cup A_3}(Illegal(\mathcal{C}_2, \mathcal{C}_3))\big)$, which is in turn included in $pre_!^*(Illegal(\mathcal{C}_1, \mathcal{C}_3))$, thanks to the previous part of the Lemma. □

COROLLARY 13 (COMPATIBILITY PRESERVATION). *Given any three modal interfaces $\mathcal{C}_i$, $i = 1...3$, such that $\mathcal{C}_2 \leq_s \mathcal{C}_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$. $\mathcal{C}_1$ compatible with $\mathcal{C}_3$ implies that $\mathcal{C}_2$ and $\mathcal{C}_3$ are also compatible.*

Proof: *This is an immediate consequence of Lemma 12. Assume $\mathcal{C}_2$ and $\mathcal{C}_3$ incompatible, meaning that $\epsilon \in \mathcal{E}_{\mathcal{C}_2 \parallel \mathcal{C}_3}$. By Lemma 12, $\epsilon = \mathbf{pr}_{A_1 \cup A_3}(\epsilon) \in \mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_3}$. Hence $\mathcal{C}_1$ and $\mathcal{C}_3$ are also incompatible.* □

Contrary to interface automata for which $\mathcal{C}_1 \parallel \mathcal{C}_2$ is a refinement of $\mathcal{C}_1 \otimes \mathcal{C}_2$ [14], relaxation of modal interfaces amounts to compute an abstraction of the product:

LEMMA 14. *Given two modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$:*

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

Proof: *Two cases are possible:*

- *if $u \in \mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2} \backslash \mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$: $must_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) = must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$ and $may_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) = may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$;*
- *if $u \in \mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ then $u \in \mathcal{L}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ and $must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u) = \emptyset$ and $may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u) = A$.*

Thus, $must_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) \supseteq must_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$ and $may_{\mathcal{C}_1 \otimes \mathcal{C}_2}(u) \subseteq may_{\mathcal{C}_1 \parallel \mathcal{C}_2}(u)$. □

Theorem 10 stated in [23, 27] now holds for the parallel composition operator.

THEOREM 15 (INDEPENDENT IMPLEMENTABILITY). *For any two modal interfaces $\mathcal{C}_1$, $\mathcal{C}_2$ and two implementations $(\mathcal{I}_1, \pi_1)$, $(\mathcal{I}_2, \pi_2)$ such that $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ and $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, it holds that $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \parallel \mathcal{C}_2$.*

Proof: *If $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ and $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, then, by Theorem 10, $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \otimes \mathcal{C}_2$.*
By the previous lemma and by the generalization of Theorem 1 in Theorem 10: $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \parallel \mathcal{C}_2$. □

5. CONCLUSION AND FUTURE WORK

This paper presents a *modal interface* framework, a unification of interface automata and modal specifications. It is a complete theory with a powerful composition algebra that includes operations such as conjunction (for requirements composition) and residuation (for component reuse but also assume/guarantee contract based reasoning [30]). However, the core contribution of the paper is a parallel composition operator that reflects a rich notion of compatibility between components, actually correcting that parallel composition proposed in [23, 27].

There are several possible directions for future research. A first step would be to implement all the concepts and operations presented in the paper and evaluate the resulting tool on concrete case studies. Extensions of modal specifications can be investigated, where states are described as valuations of a set of variables just as it has been the case for interface automata [8, 12].

Another promising direction would be a timed extension of modal interfaces. In [15], de Alfaro et al. proposed *timed interface automata* that extends timed automata just as interface automata extend finite-word automata. The semantics of a timed interface automaton is given by a timed game [13, 6], which allows to capture the *timed dimension* in composition. Up to now, composition is the only operation that has been defined on timed interface automata. In [9], Chatain et al. have proposed a notion of refinement for timed games. However monotony of parallel composition with respect to this refinement relation has not been investigated yet. In [4], *timed modal specifications* are proposed. As modal specifications, timed modal specifications admit a rich composition algebra with product, conjunction and residuation operators. Thus, a natural direction for future research would be to unify timed interface automata and timed modal specifications. This would imply a translation from timed interface automata to timed modal specifications.

Finally, we believe it is worth studying the logical expressiveness of timed modal specifications/interfaces, as it has been the case for modal specifications [18].

Acknowledgments

We are grateful to Barbara Jobstmann and Laurent Doyen who found the counter example given in Section 4.4.2.

6. REFERENCES

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[2] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proc. of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[3] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Comput. Sci.*, 11, 1980.

[4] N. Bertrand, S. Pinchinat, and J.-B. Raclet. Refinement and consistency of timed modal specifications. In *Proc. of the 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 152–163, Tarragona, Spain, 2009. Springer.

[5] S. Bliudze and J. Sifakis. A notion of glue expressiveness for component-based systems. In *Proc. of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2008.

[6] T. Brihaye, F. Laroussinie, N. Markey, and G. Oreiby. Timed concurrent game structures. In *Proc. of the 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2007.

[7] J. F. M. Burg. *Linguistic instruments in requirements engineering*. IOS Press, 1997.

[8] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. Synchronous and bidirectional component interfaces. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 414–427, 2002.

[9] Th. Chatain, A. David, and K. G. Larsen. Playing games with timed games. Research Report LSV-08-34, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2008. 15 pages.

[10] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[11] L. de Alfaro. Game models for open systems. In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 269–289. Springer, 2003.

[12] L. de Alfaro, L. D. da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable interfaces. In *5th International Workshop on Frontiers of Combining Systems (FroCos'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 81–105. Springer, 2005.

[13] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *Proc. of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003.

[14] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. of the 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'01)*, pages 109–120. ACM Press, 2001.

[15] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proc. of the 2nd Workshop on Embedded Software (EMSOFT'02)*, volume 2491 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2002.

[16] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In L. de Alfaro and J. Palsberg, editors, *Proc. of the 8th International Conference on Embedded Software (EMSOFT'08)*, pages 79–88. ACM Press, 2008.

[17] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. *Proc. of the IEEE*, 91(1):127–144, 2003.

[18] G. Feuillade. Modal specifications are a syntactic fragment of the mu-calculus. Research Report RR-5612, INRIA, June 2005.

[19] C. Fournet, C. A. R. Hoare, S. K. Rajamani, and J. Rehof. Stuck-free conformance. In *Proc. of the 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.

[20] T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *Proc. of the 14th International Symposium on Formal Methods (FM'06)*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[21] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1999.

[22] K. G. Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1989.

[23] K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O automata for interface and product line theories. In *Programming Languages and Systems, 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.

[24] K. G. Larsen, U. Nyman, and A. Wasowski. On modal refinement and consistency. In *Proc. of the 18th International Conference on Concurrency Theory (CONCUR'07)*, pages 105–119. Springer, 2007.

[25] G. Lüttgen and W. Vogler. Conjunction on processes: Full abstraction via ready-tree semantics. *Theoretical Computer Science*, 373:19–40, 2007.

[26] N. Lynch and M. R. Tuttle. An introduction to Input/Output automata. *CWI-quarterly*, 2(3), 1989.

[27] U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Aalborg University, Department of Computer Science, September 2008.

[28] J.-B. Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, december 2007. (In French).

[29] J.-B. Raclet. Residual for component specifications. In *Proc. of the 4th International Workshop on Formal Aspects of Component Software (FACS'07)*, 2007.

[30] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *Proc. of the 9th International Conference on Application of Concurrency to System Design (ACSD'09)*. IEEE Computer Society Press, 2009.