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 thorough 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

*This work was funded in part by the European IP-SPEEDS project number 033471 and the European STREP-COMBEST project number 215943.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT’09, October 12–16, 2009, Grenoble, France.
Copyright 2009 ACM 978-1-60558-627-4/09/10 ...$10.00.

General Terms
Design, Theory

Keywords
Interface Automata, Modal Specifications, Compatibility

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 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 \( S = (A, \text{must}, \text{may}) \), where \( A \) is a finite alphabet and:

\[
\text{must}, \text{may} : A^* \mapsto 2^A
\]

are partial functions satisfying the following consistency condition:

\[
\text{must}(u) \subseteq \text{may}(u). \tag{1}
\]

The fact that \( a \in \text{may}(u) \) means that action \( a \) is allowed after the trace \( u \) whereas \( a \in \text{must}(u) \) indicates that \( a \) is required after \( u \). By negation, \( a \notin \text{may}(u) \) means that \( a \) is disallowed after \( u \). The latter is often written \( a \in \text{must}(u) \).

The condition (1) naturally imposes that every required action is also allowed. We shall sometimes write \( A_S, \text{may}_S, \) and \( \text{must}_S \) to refer to the entities involved in the definition of \( S \).

When composing specifications, discrepancies between the modal informations carried out by the specifications may appear. We then consider pseudo-modal specifications, denoted \( P_S \); they are triples satisfying Definition 1 with the exception of (1). For \( P_S \) a pseudo-modal specification, a word \( u \in A^* \) is called consistently specified in \( P_S \) 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 \( P_S = (A, \text{must}, \text{may}) \) a pseudo-modal specification, the support of \( P_S \) is the least prefix-closed language \( L_{PS} \) such that: (i) \( e \in L_{PS} \), where \( e \) denotes the empty word; and (ii) \( u \in L_{PS} \) and \( a \in \text{may}(u) \) imply \( u.a \in L_{PS} \).

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 \( \text{pr}_A(v) \)) is the word over \( A \) obtained from \( v \) by erasing all symbols that do not belong to \( A \). Let \( L_v \) be a language over \( A \), the extension of \( L \) to \( C \) is the language \( L_C \) defined as:

\[
\text{pr}_A(v) \in L_C \iff v \in L \land v \in C^*.
\]

Definition 2. The shuffle product \( L_1 \times L_2 \) of two languages \( L_1 \subseteq A_1^* \) and \( L_2 \subseteq A_2^* \) is given by:

\[
L_1 \times L_2 = (L_1)^\uparrow_A \cap (L_2)^\uparrow_A, \quad \text{where} \quad A = A_1 \cup A_2.
\]

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 \( P_S = (A, \text{must}_{PS}, \text{may}_{PS}) \) be a pseudo-modal specification and let \( C \supseteq A \).

1. The weak extension of \( P_S \) to \( C \) is the pseudo-modal specification \( P_{SC} = (C, \text{must}, \text{may}) \) such that \( \forall v \in C^* : \)

\[
\begin{align*}
\text{must}(v) &= \text{must}_{PS}(\text{pr}_A(v)) \\
\text{may}(v) &= \text{may}_{PS}(\text{pr}_A(v)) \cup (C - A).
\end{align*}
\]

2. The strong extension of \( P_S \) to \( C \) is the pseudo-modal specification \( P_{SC} = (C, \text{must}, \text{may}) \) such that \( \forall v \in C^* : \)

\[
\begin{align*}
\text{must}(v) &= \text{must}_{PS}(\text{pr}_A(v)) \cup (C - A) \\
\text{may}(v) &= \text{may}_{PS}(\text{pr}_A(v)) \cup (C - A).
\end{align*}
\]

It is easy to show that \( L_{(S_{EC})} = L_{(S_{EC})} = (L_{S})_{|C} \).

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 \( P_S = (A, \text{must}, \text{may}) \) be a pseudo-modal specification.

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

2. Extended Alphabets: For \( C \supseteq A \), a prefix-closed language \( I \subseteq C^* \) is a weak implementation of \( P_S \), written \( I \models_w P_S \), iff \( I \models P_{S_C} \) holds; it is a strong implementation of \( P_S \), written \( I \models_s P_S \), iff \( I \models P_{S_C} \) holds.

Modal specifications are equivalent to the fragment of the \( \mu \)-calculus called the conjunctive \( \nu \)-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 \( I \models P_S \), then \( I \subseteq L_{PS} \) holds and every word of \( I \) is consistently specified in \( P_S \). Similarly, if \( I \models_w P_S \) or \( I \models_s P_S \), then \( I \subseteq (L_{PS})_{|C} \) holds and for every word \( v \in C^* \) of \( I \), \( \text{pr}_A(v) \) is consistently specified in \( P_S \).

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 \( P_{S_1} = (A, \text{must}_1, \text{may}_1) \) and \( P_{S_2} = (A, \text{must}_2, \text{may}_2) \) be two pseudo-modal specifications then \( P_{S_1} \) refines \( P_{S_2} \), denoted \( P_{S_1} \preceq P_{S_2} \), iff for all \( u \in L_{S_1} : \)

\[
\text{may}_1(u) \subseteq \text{may}_2(u) \quad \text{and} \quad \text{must}_1(u) \supseteq \text{must}_2(u).
\]

It can be shown that refinement is a preorder relation which implies the inclusion of supports. As a consequence, any two modal specifications \( S_1 \) and \( S_2 \) such that \( S_1 \preceq S_2 \) have equal supports \( L = L_{S_1} = L_{S_2} \) and moreover, for
all \( u \in \mathcal{L} \), \( \text{may}_1(u) = \text{may}_2(u) \) and \( \text{must}_1(u) = \text{must}_2(u) \). Thus equivalent modal specifications differ only outside of their support; a unique representant \( S = (A, \text{must}, \text{may}) \) of equivalence classes of modal specifications can be defined by assuming that for all \( u \not\in \mathcal{L}_S \), \( \text{may}(u) = \emptyset \) and \( \text{may}(u) = A \). 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 \( \rho S_1 = (A_1, \text{must}_1, \text{may}_1) \) and \( \rho S_2 = (A_2, \text{must}_2, \text{may}_2) \) be two pseudo-modal specifications with \( A_1 \supseteq A_2 \). Then \( \rho S_1 \) weakly refines \( \rho S_2 \) (which is denoted \( \rho S_1 \leq_w \rho S_2 \)), if \( \rho S_1 \leq \rho S_2 |_{\rho A_1} \), and it strongly refines \( \rho S_2 \), written \( \rho S_1 \leq_s \rho S_2 \), if \( \rho S_1 \leq \rho S_2 |_{\rho A_1} \).

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

**Theorem 2 (Consistency).** Either a pseudo-modal specification \( \rho S \) has no model, or there exists a modal specification \( \rho |S| \) having the same alphabet of actions such that \( \rho |S| \) possesses the same set of weak and strong implementations:

\[
I \models_w \rho S \Leftrightarrow I \models_w \rho |S| \\
I \models_s \rho S \Leftrightarrow I \models_s \rho |S|
\]

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

\[
S_2 \leq_w S_1 \Leftrightarrow \{ I \mid I \models_w S_2 \} \subseteq \{ I \mid I \models_w S_1 \} \\
S_2 \leq_s S_1 \Leftrightarrow \{ I \mid I \models_s S_2 \} \subseteq \{ I \mid I \models_s S_1 \}
\]

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 \( S_1 = (A_1, \text{must}_1, \text{may}_1) \) and \( S_2 = (A_2, \text{must}_2, \text{may}_2) \). 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 \( S_1 \land S_2 = \rho(S_1 \land S_2) \) where \( S_1 \land S_2 \) is defined by:

\[
\text{may}_{S_1 \land S_2}(u) = \text{may}_1(u) \cap \text{may}_2(u) \\
\text{must}_{S_1 \land S_2}(u) = \text{must}_1(u) \cup \text{must}_2(u).
\]

**Theorem 4.**

\[
I \models_w S_1 \land S_2 \Leftrightarrow I \models_w S_1 \text{ and } I \models_w S_2.
\]

The conjunction between \( S_1 \) and \( S_2 \) is exactly their greatest lower bound for the weak refinement relation: \( S_1 \land S_2 \) is the greatest specification that weakly refines both \( S_1 \) and \( S_2 \).

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 \( S = S_1 \odot S_2 \) is defined by:

\[
\text{may}_S(u) = \text{may}_1(u) \cap \text{may}_2(u) \\
\text{must}_S(u) = \text{must}_1(u) \cap \text{must}_2(u).
\]

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: \( S_1 \odot S_2 = S_1 |_{\rho A} \odot S_2 |_{\rho A} \).

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 \( S_1 \leq_s S_1 \) and \( S_2 \leq_s S_2 \), then \( S_1 \odot S_2 \leq S_1 \odot S_2 \).
2. If \( I_1 \models_s S_1 \) and \( I_2 \models_s S_2 \), then \( I_1 \times I_2 \models_s S_1 \odot S_2 \).
3. Regarding supports: \( \mathcal{L}_{S_1 \odot S_2} = \mathcal{L}_{S_1} \times \mathcal{L}_{S_2} \).

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 \( S = S_1 \parallel S_2 \) is defined by:

\[
\begin{align*}
\forall a \in \mathrm{may}_{S_2}(u) \cap \mathrm{must}_{S_2}(u) & \quad \text{if} \quad a \in \mathrm{must}_{S_1}(u) \\
\forall a \in \mathrm{must}_{S_2}(u) \cup \mathrm{may}_{S_2}(u) & \quad \text{if} \quad a \in \mathrm{must}_{S_1}(u) \\
\forall a \in \mathrm{may}_{S_2}(u) \setminus \mathrm{must}_{S_2}(u) & \quad \text{if} \quad a \in \mathrm{may}_{S_1}(u) \\
\forall a \in \mathrm{may}_{S_2}(u) \setminus \mathrm{must}_{S_2}(u) & \quad \text{if} \quad a \not\in \mathrm{must}_{S_1}(u) \\
\forall a \not\in \mathrm{may}_{S_2}(u) \cup \mathrm{must}_{S_2}(u) & \quad \text{if} \quad a \not\in \mathrm{may}_{S_1}(u) \\
\forall a \not\in \mathrm{may}_{S_2}(u) \setminus \mathrm{must}_{S_2}(u) & \quad \text{if} \quad a \not\in \mathrm{may}_{S_1}(u) \\
\end{align*}
\]

Due to the second rule, \( S_1 \parallel S_2 \) may have inconsistently specified words. As a consequence, a reduction operation may be needed and the quotient of \( S_1 \) by \( S_2 \) is \( S_1 / S_2 = \rho(S_1 / S_2) \).

For the general case of two different alphabets, the definition above is applied after an alphabet equalization step:

\[
S_1 / S_2 = S_1 \rho A / S_2 \rho A.
\]

We have the following theorems:

**Theorem 6.** Let \( S, S_1 \) and \( S_2 \) be modal specifications such that \( A S_2 \supseteq A S \supseteq A S_1 \). We have

\[
S_2 \preceq S / S_1 \Leftrightarrow S_1 \otimes S_2 \preceq S.
\]

**Theorem 7.** Let \( S, S_1 \) be modal specifications and \( I_2 \) a prefix-closed language such that \( A I_2 \supseteq A S \supseteq A S_1 \), we have

\[
I_2 \models S / S_1 \Leftrightarrow [\forall I_1 : I_1 \models S_1 \Rightarrow I_1 \otimes I_2 \models S].
\]

### 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 \( \mathcal{P} = (X, \mathcal{X}, A, \rightarrow) \), where \( X \) is the set of states, \( \mathcal{X} \) is the initial state, \( A \) is the alphabet of actions, and \( \rightarrow \subseteq X \times A \times X \) is the transition relation.

We decompose \( A = A_1 \uplus A_2 \), where \( A_1 \) is the set of inputs and \( A_2 \) is the set of outputs. In the rest of the paper, we shall often use \( a? \) to emphasize that \( a \in A_1 \) and \( a! \) for \( a \in A_2 \). 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 \( a \not\in A_1 \cap A_2 \), there exists a transition \( (x_1, y_1) \xrightarrow{a_1} (x_2, y_2) \) if 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 \( (x_1, y_1) \xrightarrow{a_1} (x_2, y_2) \) if 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 \( (x_1, y_1) \xrightarrow{a_1} (x_2, y_2) \) if 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 Illegal($P_1, P_2$) is the subset of pairs $(x_1, x_2) \in X_1 \times X_2$ such that there exists

either an action $a \in A_1 \cap A_2$ with $x_1 \xrightarrow{a} x_1' \operatorname{but not } x_2 \xrightarrow{a} x_2'$
or an action $a \in A_1 \cap A_2$ with $x_2 \xrightarrow{a} x_2'$ \operatorname{but not } x_1 \xrightarrow{a} x_1'$

where $x \xrightarrow{a} y$ means that $x \xrightarrow{a} y$ for some state $y$. If illegal states exist in the product $P_1 \times P_2$, 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 $P_1 \times P_2$ as follows. For $Y \subseteq X$, the set of states of $P_1 \times P_2$, let $\text{pre}^i(Y)$ be the subset $Z \subseteq X$ of states $x$ such that $x \xrightarrow{a} y$ for some $y \in Y$ and $a \in A_i$ \operatorname(an output action of the product). Let $\text{pre}_i^0(Y) = Y$ and, for $k \geq 0$, $\text{pre}_i^{k+1}(Y) = \text{pre}_i(\text{pre}_i^k(Y))$ and let $\text{pre}_i(Y) = \bigcup_k \text{pre}_i^k(Y)$.

The desired pruning consists in:

- Removing $\text{pre}_i^*(\text{Illegal}((P_1, P_2))$ from $X$, and
- Removing transitions to states in $\text{pre}_i^*(\text{Illegal}((P_1, P_2))$, and
- Removing unreachable states.

The result of applying the pruning to $P_1 \times P_2$ is denoted by

$P_1 || P_2$,

and is called the composition of the two interface automata. $P_1$ and $P_2$ 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 substitutability.

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

**Theorem 9 ([14]).** Let $P_1$, $P_2$, and $P_3$ be three interface automata. If $P_2$ refines $P_1$ and the set of shared actions $P_2 || P_3$ is included in the set of shared actions of $P_1 || P_3$, then $P_2 || P_3$ refines $P_1 || P_3$.

**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** $\pi : A \mapsto \{ ?, ! \}$, labeling actions with the symbols $? \operatorname(for inputs)$ or $! \operatorname(for outputs)$. We write “$a ?$, $a !”$ to express that $“\pi(a) = ?”, \operatorname{and similarly for the other case.}$ We denote by $A_? \operatorname{the set of} a \in A \operatorname{such that} \pi(a) = ?$, 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 $\pi_1$ and $\pi_2$, which is defined if $A_1 \cap A_2 = \emptyset$, is the $\pi = (A?, A!)$ such that

\[ \pi_1 \otimes \pi_2 : \left\{ \begin{array}{c} A_! = (A_1! \cup A_2!) \\ A? = (A_1? \cup A_2?) \setminus A! \end{array} \right. \]

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

**Conjunction between profiles.** The conjunction between $\pi_1$ and $\pi_2 \operatorname{denoted} \pi_1 \land \pi_2$ is the greatest lower bound of the profiles, whenever it exists. More precisely, the conjunction of profiles $\pi_1$ and $\pi_2$ 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 $\pi_1 \land \pi_2$ coincides with $\pi_1$ for every letter in $A_1$ and with $\pi_2$ 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 $A_1 \supseteq A_2 \text{ and } A_1! \supseteq A_2!$, and is then equal to the profile $\pi = (A?, A!) \operatorname{such that}$

\[ \pi_1 / \pi_2 : \left\{ \begin{array}{c} A! = A_1! \setminus A_2! \\ A? = A_1? \setminus A_2?. \end{array} \right. \]

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 $\mathcal{C} = (S, \pi)$, where $S$ is a modal specification on the alphabet $A_S$ and $\pi : A_S \mapsto \{ ?, ! \}$ is a profile.
A model for a modal interface is a tuple \((I, \pi')\), where \(I\) is a prefix-closed language and \(\pi'\) is a profile for \(I\). We say that \((I, \pi')\) strongly implements \((S, \pi)\), written \((I, \pi') \models (S, \pi)\), if \(I \models S\) and \(\pi' \leq \pi\), and similarly for weak implementation. We say that \((S_1, \pi_1) \leq (S_2, \pi_2)\) if \(S_2 \subseteq S_1\) 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 \(C_1 = (S_1, \pi_1)\) and \(C_2 = (S_2, \pi_2)\), and let \(* \in \{\wedge, \odot, /\}\). If \(\pi_1 * \pi_2\) is defined, then

\[
C_1 \ast C_2 = (S_1 \ast S_2, \pi_1 \ast \pi_2).
\]

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 \(P = (X, x_0, A, \rightarrow)\). We assume \(P\) to be deterministic and we let \(L_P\) denote the (prefix-closed) language defined by \(P\). The alphabet of \(S_P\) is \(A_{SP} = A\) and modalities are defined for all \(u \in A_{SP}\):

\[
\begin{align*}
& a? \in \text{must}_{SP}(u) & \text{if} & u.a? \in L_P \\
& a! \in \text{may}_{SP}(u) \setminus \text{must}_{SP}(u) & \text{if} & u.a! \in L_P \\
& a? \in \text{may}_{SP}(u) \setminus \text{must}_{SP}(u) & \text{if} & u \in L_P \\
& a! \notin \text{may}_{SP}(u) & \text{and} & u.a? \notin L_P, \quad u.a! \notin L_P \quad (4) \\
& a \in \text{may}_{SP}(u) \setminus \text{must}_{SP}(u) & \text{if} & u \notin L_P.
\end{align*}
\]

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:

\[
L_{SP} = L_P \uplus \{u.a?, v \mid u \in L_P, u.a? \notin L_P, v \in A^*_P\}. \quad (5)
\]

It is worth making some comments about this translation, given by formulas (4,5). Regarding formula (5), the supporting language \(L_{SP}\) allows the environment to violate the constraints set on it by the interface automaton \(P\). 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, \(L_{SP}\) 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 \(C_1 = (S_1, \pi_1)\) and \(C_2 = (S_2, \pi_2)\) with \(S_1\) defined over \(A_1\) and \(S_2\) defined over \(A_2\). We assume that \(C_1\) and \(C_2\) do not share common output actions (which is the composability requirement similar to the one for interface automata). We first compute the product between \(C_1\) and \(C_2\) following Definition 4.3.

We then define \(\text{Illegal}(C_1, C_2)\) to be the subset of words \(u\) belonging to the support of \(C_1 \odot C_2\), such that there exists either an action \(a \in A_1 \cap A_2\),

\[
\text{with } a \in \text{may}_1(u_1) \setminus \text{must}_2(u_2)
\]

or an action \(a \in A_1 \cap A_2\),

\[
\text{with } a \in \text{may}_2(u_2) \setminus \text{must}_1(u_1).
\]

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

\[
\text{pre}_C(U) = \{v \in L_C \mid \exists! v.a! \in \text{may}(v), v.a! \in U\}.
\]

Let \(\text{pre}_C^0(U) = U\), and, for \(k \geq 0\),

\[
\text{pre}_C^{k+1}(U) = \text{pre}_C(\text{pre}_C^k(U)).
\]

Finally, let \(\text{pre}_C^\ast(U) = \bigcup_{k \geq 0} \text{pre}_C^k(U)\).

The composition of two modal interfaces is obtained from their product by removing states in \(\text{pre}_C^\ast(U)\), 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 \(C_1\) and \(C_2\) is denoted \(C_1 \parallel C_2\).

Theorem 10 in [23, 27] says that

"(Independent Implementability). For any two composable modal interfaces \(C_1, C_2\) and two implementations \((I_1, \pi_1)\) and \((I_2, \pi_2)\), if \((I_1, \pi_1) \leq C_1\) and \((I_2, \pi_2) \leq C_2\), then it holds that \((I_1, \pi_1) \times (I_2, \pi_2) \leq C_1 \parallel C_2\)."

The following example shows that Theorem 10 in [23, 27] is wrong.
Whenever a word in \( L_{C_1 \otimes C_2} \) 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 \( C_1 \) and \( C_2 \) is the language \( E_{C_1 \parallel C_2} = \text{pre}^* \left( \text{Illegal}(C_1, C_2) \right) \). Modal interfaces \( C_1 \) and \( C_2 \) are said to be compatible if and only if the empty word \( \epsilon \) is not in \( E_{C_1 \parallel C_2} \).

**Definition 12 (parallel composition).** Given two modal interfaces \( C_1 \) and \( C_2 \), the relaxation of \( C_1 \otimes C_2 \) is obtained by applying the following pseudo-algorithm to \( C_1 \otimes C_2 \):

- for all \( v \) in \( L_{C_1 \otimes C_2} \)
- for all \( a \) in \( A \)
- if \( v \notin E_{C_1 \parallel C_2} \) and \( v,a \in E_{C_1 \parallel C_2} \)
- then:
- for all \( w \) in \( A^* \)
- must(\( v.a.w \)) := \emptyset
- may(\( v.a.w \)) := \( A \)
- end for
- end if
- end for

If \( C_1 \) and \( C_2 \) are compatible, the relaxation of \( C_1 \otimes C_2 \) is called the parallel composition of \( C_1 \) and \( C_2 \), denoted by \( C_1 \parallel C_2 \). Whenever \( C_1 \) and \( C_2 \) are incompatible, the parallel composition \( C_1 \parallel C_2 \) 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 \((I_1, I_2)\) of strong implementations of \( C_1 \) and \( C_2 \). If this occurs, then \( C_1 \parallel C_2 \) 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 \( [C_1 \parallel C_2]_0 \), with \( A = \{a!\, b!, c?\} \), which has \( I_1 \times I_2 \) 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 \( C_i, i = 1...3 \), such that \( C_2 \leq C_1 \) and \( A_1 \cap A_2 \supseteq A_2 \cap A_3 \):

- \( \text{pr}_{A_1 \cup A_3}(\text{Illegal}(C_2, C_3)) \) is included in \( \text{Illegal}(C_1, C_3) \);
- \( \text{pr}_{A_1 \cup A_3}(E_{C_2 \parallel C_3}) \) is included in \( E_{C_1 \parallel C_3} \).

**Proof:** Consider an illegal word \( u \in \text{Illegal}(C_2, C_3) \). This means that there exists an action \( a \in A_2 \cap A_3 \) such that (i) either \( a \) is an output of \( C_2 \) and an input of \( C_3 \), such that \( a \in \text{may}_2(\text{pr}_{A_3}(u)) \) and \( a \notin \text{must}_3(\text{pr}_{A_3}(u)) \), or (ii) \( a \) is an input of \( C_2 \) and an output of \( C_3 \), such that \( a \notin \text{must}_2(\text{pr}_{A_3}(u)) \) and \( a \in \text{may}_3(\text{pr}_{A_3}(u)) \).

By Definition 6, \( u \) is also in \( L_{C_1 \otimes C_1} \). By Definition 3, \( u' = \text{pr}_{A_1 \cup A_3}(u) \) belongs to \( L_{C_1 \otimes C_1} \).

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 \( C_1 \) and an input of \( C_3 \), such that \( a \in \text{may}_1(\text{pr}_{A_3}(u')) \) and \( a \notin \text{must}_3(\text{pr}_{A_3}(u')) \), or (ii) \( a \) is an input of \( C_1 \) and an output of \( C_3 \), such that \( a \notin \text{must}_1(\text{pr}_{A_3}(u')) \) and \( a \in \text{may}_3(\text{pr}_{A_3}(u')) \).

Meaning that \( u' \in \text{Illegal}(C_1, C_3) \), 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 \( \text{pr}_{A_1 \cup A_3}(\text{Illegal}(C_2, C_3)) \) is included in the set \( \text{pr}_{A_2 \cup A_3}(\text{Illegal}(C_2, C_3)) \), which is in turn included in \( \text{pr}_{A_2 \cup A_3}(\text{Illegal}(C_1, C_3)) \), thanks to the previous part of the Lemma. QED.
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].

6. REFERENCES


Acknowledgments

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


