A Modal Interface Theory for Component-based Design

Jean-Baptiste Raclet†
INRIA Grenoble - Rhônes-Alpes, France
Jean-Baptiste.Raclet@irit.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
axel.legay@irisa.fr

Roberto Passerone
University of Trento, Italy
roberto.passerone@unitn.it

Abstract. This paper presents the modal interface theory, a unification of interface automata and modal specifications, two radically dissimilar models for interface theories. Interface automata is a game-based model, which allows the designer to express assumptions on the environment and which uses an optimistic view of composition: two components can be composed if there is an environment where they can work together. Modal specifications are a language theoretic account of a fragment of the modal mu-calculus logic with a rich composition algebra which meets certain methodological requirements but which does not allow the environment and the component to be distinguished. The present paper contributes a more thorough unification of the two theories by correcting a first attempt in this direction by Larsen et al., drawing a complete picture of the modal interface algebra, and pushing the comparison between interface automata, modal automata and modal interfaces even further.

The work reported here is based on earlier work presented in [41] and [42].

Keywords: Component-based System, Compositional Reasoning, Interface Theory, Interface Automata, Modal Specifications.

*This work was funded in part by the European IP-SPEEDS project number 033471 and the European STREP-COMBEST project number 215543.
†Address for correspondence: INRIA Grenoble - Rhônes-Alpes, France
1. **Introduction**

Nowadays, systems are tremendously large 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, the study of mathematical foundations that allow designers to reason at the abstract level of interfaces is a very active research area. According to our understanding of industrial needs (see [5] for a discussion), 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 substitutability.* Refinement allows one to replace, in any context, an interface by a more detailed version of it. Refinement should entail substitutability 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 interface that refines two different interfaces. This is called *shared refinement* [22]. In many situations, we are looking for the *greatest lower bound*, i.e., the shared refinement that could be refined by any other shared refinement.

3. *Interfaces are closed under conjunction.* Large systems are concurrently developed for their different aspects or viewpoints by different teams using different frameworks and tools. Examples of such aspects include the functional aspect, the safety or reliability aspect, the timing aspect. Each of these aspects requires specific frameworks and tools for their analysis and design. Yet, they are not totally independent but rather interact. The issue of dealing with multiple aspects or multiple viewpoints is thus essential. This implies that several introductions are associated with a same system, sub-system, or component, namely (at least) one per viewpoint. These introductions are to be interpreted in a conjunctive way. The need for supporting conjunctive introductions also follows from the current practice in which early requirement capture relies on Doors or even Excel sheets collecting many individual requirements. The latter typically consist of English text, semi-formal languages whose sentences are translatable into predefined behavioral patterns, or even graphical scenario languages.

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. *Interface quotient supports incremental design and component reuse.* A quotenting 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.
6. **A verification procedure.** In addition to the fact that an interface already represents a set of properties, one should be able to verify if an interface satisfies a set of requirements written in some specification language.

7. **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.

Building good interface theories has been the subject of intensive studies (see, e.g., [29, 20, 11, 23, 25, 18, 21]). In this paper we will concentrate on two models: (1) *interface automata* [20] and (2) *modal specifications* [30]. Interface automata is a game-based variation of input/output automata which deals with open systems, their refinement and composition, and puts the emphasis on interface compatibility. Modal specifications is a language-theoretic account of a fragment of the modal mu-calculus logic [24] which admits a richer composition algebra with product, conjunction and residuation operators.

Modal specifications correspond to deterministic modal automata [30], 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/transition systems.

Satisfiability of modal specifications is decidable. Refinement between modal specifications coincides with model inclusion. Conjunction is effectively computed via a product-like construction. It can be shown that the conjunction of two modal specifications corresponds to their greatest common refinement. Combination of modal specifications, handling synchronization products *à la* Arnold and Nivat [4], and the dual quotient combinators can be efficiently handled in this setting [39, 40].

In interface automata [20], an interface is represented by an input/output automaton [34], 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.

Refinement between interface automata corresponds to the alternating refinement relation between games [2], i.e., an interface refines another if its environment is more permissive whereas its component is more restrictive. Shared refinement is defined in an ad-hoc manner [22] for a particular class of interfaces [13]. Contrary to most interface 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 to as compatibility of interfaces. A quotient, which is the adjoint of the game-based composition, has been proposed in [10] for the deterministic case.

It is worth mentioning that, in existing work on interface automata and modal specifications, there is nothing about dissimilar alphabets. This is somehow surprising as it seems to be a quite natural question when performing operations that involve several components, e.g., conjunction, composition, and quotient. As we shall see in this paper, an explicit mechanism to handle dissimilar alphabets is
not needed when considering interface automata, since conjunction is not discussed for this model. For the case of composition/quotient, instead, we shall see that the notion is implicitly encompassed in the definition of compatibility. Conjunction and quotient operators [30, 39, 40] that have been proposed for modal specifications do not take dissimilar alphabets into account. One thus needs to extend those operators to this more general setting. This is one of the subjects of this paper.

In conclusion, both models have advantages and disadvantages:

- Interface automata is a model that allows designers to make assumptions on the environment, which is mainly useful to derive a rich notion for composition with compatibility issues. In addition, the notion of dissimilar alphabets is not needed. Unfortunately, the model is incomplete as conjunction and shared refinement are not defined.

- Modal specification is a rich language-algebraic model on which most of the 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. Moreover, the notion of dissimilar alphabets is missing.

It is thus worth considering unifying the frameworks of interface automata and modal specifications. A first attempt was made by Larsen et al. [31, 36] who considered modal interfaces, which are modal specifications whose actions are also typed in input or output attributes. A modal interface can be viewed as simply a modal specification except for the composition operation for which the modalities are an additional complication. Refinement for modal interfaces is the same as refinement for modal specifications, while composition is the one from interface automata. Larsen et al. have shown that refinement for modal specifications is compatible with the composition operation for interface automata [31, 36]. The main problem with their results is that the composition operator is incorrect. Indeed, contrary to what is claimed by the authors, their composition operator is not monotone with respect to satisfaction. This fails to ensure that two compatible interfaces may be implemented separately. Moreover, requirements such as dissimilar alphabets, conjunction, and component reuse are not considered.

The present paper adds a new stone to the cathedral of results on interface theories by (1) proposing a new theory for dissimilar alphabets, (2) correcting the modal interface composition operator presented in [31, 36], (3) pushing the comparison between interface automata, modal automata and modal specifications and modal interfaces further, and (4) reasoning on architectural design for component-based systems.

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 [31, 36]. Section 5 is dedicated to architectural design. Finally, in Section 6, we draw our conclusion and discuss future extensions for the model of modal interfaces.

2. Modal specifications

This section starts with an overview of existing results developed in [30, 39, 40] for modal specifications defined over a global alphabet (Sections 2.1, 2.2 and 2.3). We also propose a new methodology to encompass dissimilar alphabets (Section 2.4).
### 2.1. The Framework

Following our previous work [39, 40, 42], we will define modal specifications in terms of languages, knowing that they can also be represented by deterministic automata whose transitions are typed with *may* and *must* modalities [30]. We start with the following definition.

**Definition 2.1. (Modal specification)**

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:

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

If $a \in \text{may}(u)$, then $a$ 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 $a \in \text{mustnot}(u)$. Condition (1) naturally imposes that every required action is also allowed. We shall sometimes write $\text{may}_{S_i}$ and $\text{must}_{S_i}$ (or $\text{may}_i$ and $\text{must}_i$ for short) to refer to the entities involved in the definition of $S_i$.

Modal specifications that generate regular languages can be represented by deterministic modal automata, i.e., deterministic finite-word automata with two types of transitions: solid transitions if the action is required in the source state and dashed transitions if it is allowed but not required. The concept is illustrated with the example.

**Example 2.1.** Consider a producer whose alphabet of actions includes `msg` for when the producer sends a message as well as two kinds of acknowledgment for transmission: `ack` in case of success and `nack` in case of failure. Assume also the existence of an action `extra` which occurs when extra resources are requested to dispatch a message.

A functional specification $Fun$ for the producer is given in Figure 1(a). It specifies that a `msg` may be sent again. Moreover every `msg` may be acknowledged. Additionally, the producer may request extra resources at any moment.

When composing specifications, discrepancies between the modal information carried out by the specifications may appear. We then consider pseudo-modal specifications (also called mixed transition systems in [16]), denoted $\mathcal{P}S$; they are triples satisfying Definition 2.1 with the exception of (1). For
A pseudo-modal specification, a word \( u \in A^* \) is called consistently specified in \( pS \) if it satisfies (1) and inconsistent otherwise; modal specifications correspond exactly to the subclass of consistent pseudo-modal specifications, that is pseudo-modal specifications such that every \( u \in A^* \) is consistently specified.

For \( pS = (A, \text{must}, \text{may}) \) a pseudo-modal specification, the support of \( pS \) is the least prefix-closed language \( L_{pS} \) such that

(i) \( \epsilon \in L_{pS} \), where \( \epsilon \) denotes the empty word; and

(ii) \( u \in L_{pS} \) and \( a \in \text{may}(u) \) imply \( u.a \in L_{pS} \).

### 2.2. Implementation, refinement and consistency

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

**Definition 2.2. (implementation)**

A prefix-closed language \( I \subseteq A^* \) is an implementation (or model) of a pseudo-modal specification \( pS = (A, \text{must}, \text{may}) \), denoted by \( I \models pS \), if

\[
\forall u \in I \Rightarrow \text{must}(u) \subseteq I_u \subseteq \text{may}(u)
\]

where \( I_u \) is the set of actions \( a \in A \) such that \( u.a \in I \).

**Example 2.2.** A model for the specification given in Figure 1(a) is presented in Figure 1(b). It indicates that every message will be acknowledged either positively or negatively. Moreover, an extra resource is requested if the message has to be re-emitted.

**Lemma 2.1.** Let \( I \subseteq A^* \) be a prefix-closed language and \( pS \) a pseudo-modal specification over \( A \). If \( I \models pS \), then \( I \subseteq L_{pS} \) holds and every word of \( I \) is consistently specified in \( pS \).

The concept of thorough refinement follows immediately from Definition 2.2 by comparing, through set inclusion, the sets of implementations associated to two specifications.

**Definition 2.3. (thorough refinement)**

There exists a thorough refinement between specification \( pS_1 \) and specification \( pS_2 \) if and only if any model of \( pS_1 \) is also a model of \( pS_2 \).

Thorough refinement has been extensively studied in [32] and compared to the more syntactic notion of modal refinement that is recalled hereafter.

**Definition 2.4. (modal refinement)**

Let \( pS_1 \) and \( pS_2 \) be two pseudo-modal specifications. The specification \( pS_1 \) refines \( pS_2 \), written \( pS_1 \leq pS_2 \), if and only if, for all \( u \in L_1 \), \( \text{may}_1(u) \subseteq \text{may}_2(u) \) and \( \text{must}_1(u) \supseteq \text{must}_2(u) \).
It is easy to see that modal refinement is a preorder relation that implies inclusion of supports:

\[ \rho S_1 \leq \rho S_2 \implies L_{\rho S_1} \subseteq L_{\rho S_2} \]

Any two modal specifications \( S_1 \) and \( S_2 \) such that \( S_1 \leq S_2 \leq S_1 \) have equal supports \( L = L_{\rho S_1} = L_{\rho S_2} \) and for all \( u \in L \), \( \text{may}_1(u) = \text{may}_2(u) \) and \( \text{must}_1(u) = \text{must}_2(u) \). Said differently, equivalent modal specifications differ only outside of their support. A unique representative of equivalence classes of modal specifications is defined by assuming that for all \( u \notin L_S \), \( \text{must}(u) = \emptyset \) and \( \text{may}(u) = A \). In the sequel, only modal specifications satisfying this property are considered. Under this assumption, modal refinement is a partial order relation on modal specifications.

In [39, 40, 6], it is shown that modal refinement for modal specifications is sound and complete, i.e., it is equivalent to thorough refinement. For nondeterministic modal specifications, checking thorough refinement is PSPACE-hard [3] (and also EXPTIME). As modal refinement is P-complete, a faster decision procedure exists in the deterministic case.

The following result relates implementations to consistency, for a pseudo-modal specification.

**Theorem 2.1. (consistency [39, 40])**

Let \( \rho S \) be a pseudo-modal specification. Either \( \rho S \) possesses no implementation, or there exists a largest (for refinement order) modal specification \( \rho(\rho S) \) having the same alphabet of actions and such that \( \rho(\rho S) \leq \rho S \). In addition, \( \rho(\rho S) \) possesses the same set of implementations as \( \rho S \).

The modal specification \( \rho(\rho S) \) is called the pruning of \( \rho S \). It is obtained from \( \rho S \) through the following steps:

1. Start from \( R_0 \), a copy of \( \rho S \);

2. Let \( U_0 \) be the set of words inconsistently specified in \( R_0 \), meaning that \( u \in U_0 \) does not satisfy condition (1). For each \( u \in U_0 \), set \( \text{may}_{R_0}(u) = A \) and \( \text{must}_{R_0}(u) = \emptyset \). Then, for each word \( v \in A^* \) such that \( v.a = u \) for some \( u \in U_0 \) and \( a \in A \), remove \( a \) from \( \text{may}_{R_0}(v) \). Performing these two operations yields a pseudo-modal specification \( R_1 \) such that \( U_0 \) is consistently specified in \( R_1 \). Since we have only removed inconsistently specified words from \( L_{R_0} \), by Lemma 2.1, \( R_1 \) and \( R_0 \) possess identical sets of implementations.

3. Observe that, if \( a \in \text{must}_{R_1}(v) \), then \( v \) becomes inconsistently specified in \( R_1 \). So we repeat the above step on \( R_1 \), by considering \( U_1 \), the set of words \( u \) inconsistently specified in \( R_1 \). Let \( \Delta_1 \subseteq U_0 \times U_1 \) be the relation consisting of the pairs \( (u, v) \) such that \( v.a = u \) for some \( a \) and \( v \) is inconsistently specified in \( R_1 \). Note that \( v \) is a strict prefix of \( u \).

4. Repeating this, we get a sequence of triples \( (R_k, U_k, \Delta_k)_{k \geq 0} \) such that 1) \( \bigcup_{m \leq k} U_m \) is consistently specified in \( R_{k+1} \), and 2) \( \text{may}_{R_{k+1}}(v) \subseteq \text{may}_{R_k}(v) \) for each \( v \), with strict inclusion whenever \( v.a = u \) for some \( u \in U_k \), and 3) \( \Delta_{k+1} \subseteq U_k \times U_{k+1} \) is the relation consisting of the pairs \( (u, v) \) such that \( v.a = u \) for some \( a \) and \( v \) is inconsistently specified in \( R_{k+1} \) again, \( v \) is a strict prefix of \( u \).

\(^1\)Completeness of modal refinement does not hold for nondeterministic modal automata [32]. It holds in our case since we work with specifications (for which determinism is hardwired).
5. Call chain a sequence $u_0, u_1, \ldots$ of words such that $(u_k, u_{k+1}) \in \Delta_{k+1}$ for every $k \geq 0$. Since $u_{k+1}$ is a strict prefix of $u_k$, every chain is of length at most $|u_0|$. Thus, every inconsistently specified word of $pS$ is removed after finitely many steps of the above algorithm. This proves that the procedure eventually converges. The limit $\rho(pS)$ is consistent and is given by:

\[
\begin{align*}
\text{may}(u) &= \bigcap_k \text{may}_{R_k}(u) \\
\text{must}(u) &= \begin{cases} \\
\text{must}_{pS}(u) & \text{if } \text{must}_{pS}(u) \subseteq \text{may}(u) \\
\emptyset & \text{otherwise}
\end{cases}
\end{align*}
\]

The above procedure terminates in finitely many steps if the support of the pseudo-modal specification is regular which is, in particular, the case of pseudo-modal specifications originated from a deterministic pseudo-modal automaton. This procedure also entails a sufficient condition for the satisfiability problem: a pseudo-modal specification admits a model if and only if there is no word $u \in \mathcal{L}_{pS}$ such that $u$ is inconsistent and for all prefixes $v$ of $u$ if $u = v.a.v'$ then $a \in \text{must}(v)$. Hence this problem is NLOGSPACE-complete; it is PSPACE-hard for nondeterministic pseudo-modal specifications [3].

### 2.3. Operations on modal specifications

**Greatest Lower Bound:** The set of all pseudo-modal specifications equipped with modal refinement $\leq$ is a lattice. We denote by $pS_1 \& pS_2$ the Greatest Lower Bound (GLB) of $pS_1$ and $pS_2$ defined over the same alphabet. The GLB $pS_1 \& pS_2$ can be computed as

\[
\begin{align*}
\text{may}(u) &= \text{may}_1(u) \cap \text{may}_2(u) \\
\text{must}(u) &= \text{must}_1(u) \cup \text{must}_2(u)
\end{align*}
\]

Observe that, even if $pS_1$ and $pS_2$ satisfy (1), it is not guaranteed that $pS_1 \& pS_2$ does too. Hence, by using Theorem 2.1, for $S_1$ and $S_2$ two modal specifications, we define $S_1 \land S_2$ as being the (uniquely defined) modal specification

\[
S_1 \land S_2 = \rho(S_1 \& S_2).
\]

GLB satisfies the following key property, which relates it to logic formulas:

**Theorem 2.2. (conjunctive interfaces [39, 40])**

Let $\mathcal{I}$ be a prefix-closed language and $S_1$ and $S_2$ be modal specifications. Then,

\[
\mathcal{I} \models S_1 \land S_2 \iff \mathcal{I} \models S_1 \text{ and } \mathcal{I} \models S_2
\]

The following holds regarding supports:

\[
\mathcal{L}_{S_1 \land S_2} \subseteq \mathcal{L}_{S_1} \cap \mathcal{L}_{S_2}, \text{ with equality if and only if no pruning is needed, i.e., } S_1 \land S_2 = S_1 \& S_2.
\]

**Composition:** Let $S_1$ and $S_2$ be two modal specifications over the same alphabet. Their composition $S_1 \otimes S_2$ is defined by

\[
\begin{align*}
\text{may}(u) &= \text{may}_1(u) \cap \text{may}_2(u) \\
\text{must}(u) &= \text{must}_1(u) \cap \text{must}_2(u)
\end{align*}
\]

The following theorem shows that composition ensures substitutability.
Theorem 2.3. (substitutability in composition [39, 40])

Let $I_1, I_2$ be two prefix-closed languages and $S_1, S_2, S'_1$ and $S'_2$ be modal specifications:

1. If $S'_1 \leq S_1$ and $S'_2 \leq S_2$, then $S'_1 \otimes S'_2 \leq S_1 \otimes S_2$.

2. If $I_1 \models S_1$ and $I_2 \models S_2$, then $I_1 \times I_2 \models S_1 \otimes S_2$, where $I_1 \times I_2 = I_1 \cap I_2$.

3. The following holds regarding supports: $L_{S_1 \otimes S_2} = L_{S_1} \cap L_{S_2}$.

Residuation: We now discuss the residuation operation which was introduced in [39, 40]. We will show that this operation is the adjoint of composition. For $S_1$ and $S_2$ two modal specifications, we first define their pseudo-quotient $S_1 / / S_2$ according to the following disjunctive and exhaustive cases:

\[
\begin{align*}
  a \in \text{may}(u) \cap \text{must}(u) & \quad \text{if} \quad a \in \text{must}_1(u) \\
  \quad \text{and} \quad a \in \text{must}_2(u) \\
  a \in \text{must}(u) \setminus \text{may}(u) & \quad \text{if} \quad a \in \text{must}_1(u) \\
  \quad \text{and} \quad a \notin \text{must}_2(u) \\
  a \in \text{may}(u) \setminus \text{must}(u) & \quad \text{if} \quad a \in \text{may}_1(u) \\
  \quad \text{and} \quad a \notin \text{must}_1(u) \\
  a \in \text{may}(u) \setminus \text{must}(u) & \quad \text{if} \quad a \notin \text{may}_1(u) \\
  \quad \text{and} \quad a \notin \text{must}_2(u) \\
  a \notin \text{may}(u) \cup \text{must}(u) & \quad \text{if} \quad a \notin \text{may}_1(u) \\
  \quad \text{and} \quad a \notin \text{may}_2(u)
\end{align*}
\]

Observe that, due to the second case, $S_1 / / S_2$ is not consistent. Having defined $S_1 / / S_2$, using the pruning operation of Theorem 2.1, we can now set

\[ S_1 / S_2 = \rho(S_1 / / S_2). \quad (5) \]

Any prefix-closed language $I \subseteq A^*$ can be viewed as a modal specification whose must set coincides with its may set: $\forall u \in A^*, \text{must}(u) = \text{may}(u) = I_u$. Using this embedding, the quotient of two prefix-closed languages can be defined. Observe that, because of the fourth rule, the quotient of two languages is a modal specification that is not necessarily a language.

We now show that the quotient operation is indeed the adjoint of the composition operation:

Theorem 2.4. (residuation [39, 40])

Let $I_1, I_2$ be prefix-closed languages and $S_1, S_2, S$ be modal specifications. Then,

1. $S_1 \otimes S_2 \leq S$ if and only if $S_2 \leq S / S_1$

2. $\forall I : [I_1 \models S_1 \Rightarrow I_1 \times I_2 \models S] \iff I_2 \models S / S_1$.

Example 2.3. Quotient and conjunction are illustrated in Figure 2. Suppose one aims at realizing a system whose behavior is given by the left-hand side specification: every message must be acknowledged...
positively. For this purpose, a preexisting component conforming to the middle-hand side specifications is available in the context; it implements the specification \( \text{Fun} \) of Figure 1(a) with the additional assumption that the communication channel never distributes a negative acknowledgment. Then, the product of the context with any implementation of the right-hand side specification is guaranteed to be an implementation of the desired behavior.

### 2.4. Dissimilar alphabets

Complex systems are built by composing and combining many subsystems or components. Clearly, those objects should possess their own local alphabet of ports and variables. Dealing with those local aspects when developing the fundamental services seems like a trivial notice but has deep technical consequences. As we shall see in this section, modalities appear as an elegant solution to address alphabet equalization with appropriate flexibility.

Let us first recall how alphabet equalization is performed for the shuffle product of languages. For \( w \) a word over some alphabet \( A \), and \( B \subseteq A \), let \( \text{pr}_B(w) \) denote the word over \( B \) obtained by erasing, from \( w \), all symbols not belonging to \( B \). For \( \mathcal{L} \) a language over \( A \) and \( B \subseteq A \subseteq C \), the restriction of \( \mathcal{L} \) to \( B \) is the language \( \mathcal{L}_{\downarrow B} = \{ u \in B^* \mid u = \text{pr}_B(w), w \in \mathcal{L} \} \) and the extension of \( \mathcal{L} \) to \( C \) is the language \( \mathcal{L}_{\uparrow C} = \{ u \in C^* \mid \text{pr}_A(u) \in \mathcal{L} \} \). The shuffle product \( \mathcal{L}_1 \times \mathcal{L}_2 \) of the two languages \( \mathcal{L}_1 \subseteq A_1^* \) and \( \mathcal{L}_2 \subseteq A_2^* \) is then defined as

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

The shuffle product uses inverse projection to equalize alphabets. The same holds for automata over dissimilar alphabets and their synchronous product.

Using modalities allows for a *neutral* procedure for equalizing alphabets. The principle is as follows.
Observe that, by (4),

\[ a \in \text{must}_1(s) \quad \text{and} \quad a \in \text{whatever}_2(s) \]

\[ \Downarrow \]

\[ a \in \text{whatever}(s) \]

holds if the two interfaces are combined using parallel composition (here, \text{whatever} denotes an arbitrary modality). Similarly, by (2),

\[ a \in \text{may}_1(s) \quad \text{and} \quad a \in \text{whatever}_2(s) \]

\[ \Downarrow \]

\[ a \in \text{whatever}(s) \]

holds if the two interfaces are combined using conjunction. The observation above reveals our solution: alphabet extension is performed by setting the specific modalities for extended traces, specifically

- \text{may} in case of the conjunction \( \land \);
- \text{must} in case of the parallel composition \( \otimes \).

These two types of alphabet extensions are called \textit{weak} and \textit{strong}. This is a key contribution of our work as it will provide us with a very elegant way of dealing with dissimilar alphabets.

**Definition 2.5. (weak and strong extensions)**

Let \( \mathcal{S} = (A, \text{must}_S, \text{may}_S) \) be a pseudo-modal specification and let \( C \supseteq A \).

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

\[
\begin{align*}
\text{must}(v) &= \text{must}_S(\text{pr}_A(v)) \\
\text{may}(v) &= \text{may}_S(\text{pr}_A(v)) \cup (C \setminus A)
\end{align*}
\]

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

\[
\begin{align*}
\text{must}(v) &= \text{must}_S(\text{pr}_A(v)) \cup (C \setminus A) \\
\text{may}(v) &= \text{may}_S(\text{pr}_A(v)) \cup (C \setminus A)
\end{align*}
\]

Regarding supports, the following equalities hold: \( \mathcal{L}(\mathcal{S}\uparrow C) = \mathcal{L}(\mathcal{S}\uparrow C) = (\mathcal{L}_C)_{\uparrow C} \). We are now ready to extend the operations of Sections 2.2 and 2.3 to the case of dissimilar alphabets.

**Definition 2.6.** Let \( \mathcal{S}, \mathcal{S}_1 \) and \( \mathcal{S}_i \) be pseudo-modal or modal specifications over alphabets \( A, A_i \) for \( i = 1, 2 \), respectively. The relations and operations of Section 2.2 are redefined as follows:

- [weak implementation; \( C \supseteq A \)]

\[ \mathcal{I} \subseteq C^* \models_w \mathcal{S} \iff \mathcal{I} \models \mathcal{S}\uparrow C \]

- [strong implementation; \( C \supseteq A \)]

\[ \mathcal{I} \subseteq C^* \models_s \mathcal{S} \iff \mathcal{I} \models \mathcal{S}\uparrow C \]
\[ A_2 \supseteq A_1 \Rightarrow S_2 \preceq w S_1 \text{ iff } S_2 \preceq S_{1\uparrow A_2} \]

\[ A_2 \supseteq A_1 \Rightarrow S_2 \preceq s S_1 \text{ iff } S_2 \preceq S_{1\uparrow A_2} \]

\[ (\text{operators; } C = A_1 \cup A_2) \]

\[ S_1 \land S_2 = S_{1\uparrow C} \land S_{2\uparrow C} \]

\[ S_1 \otimes S_2 = S_{1\uparrow C} \otimes S_{2\uparrow C} \]

\[ S_1 / S_2 = S_{1\uparrow C} / S_{2\uparrow C} \]

Note the careful use of weak and strong extensions in the different operations. The results of Sections 2.2 and 2.3 are slightly weakened as indicated next.

**Theorem 2.5.** Let \( S, S_i \) and \( S_i' \) be modal specifications defined over \( A, A_i \) and \( A_i' \) respectively, for \( i = 1, 2 \).

1. Weak and strong implementation/refinement relations are related as follows:

\[ \models s \subseteq \models w \text{ and } \preceq_s \subseteq \preceq_w \]

2. Weak and strong modal refinement are both sound and complete w.r.t. weak and strong thorough refinement, respectively:

\[ S_2 \preceq w S_1 \iff \{ I \mid I \models w S_2 \} \subseteq \{ I \mid I \models w S_1 \} \]

\[ S_2 \preceq s S_1 \iff \{ I \mid I \models s S_2 \} \subseteq \{ I \mid I \models s S_1 \} \]

3. The following holds regarding conjunction:

\[ I \models w S_1 \land S_2 \iff I \models w S_1 \text{ and } I \models w S_2 \]

4. Theorem 2.3 still holds when alphabets are different, provided that strong refinement and implementation are used — it is actually false if weak refinement or implementation are used:

- If \( S_1' \preceq s S_1 \) and \( S_2' \preceq s S_2 \), then \( S_1' \otimes S_2' \preceq s S_1 \otimes S_2 \);
- If \( I_1 \models s S_1 \) and \( I_2 \models s S_2 \), then \( I_1 \times I_2 \models s S_1 \otimes S_2 \);
- \( S_1' \preceq w S_1 \) and \( S_2' \preceq w S_2 \) in general do not imply that \( S_1' \otimes S_2' \preceq w S_1 \otimes S_2 \);
- \( I_1 \models w S_1 \) and \( I_2 \models w S_2 \) in general do not imply that \( I_1 \times I_2 \models w S_1 \otimes S_2 \).
5. Relations between the quotient and the composition operators are preserved provided additional assumptions on alphabets:

\[
\begin{align*}
S_2 \leq_S S/S_1 \\
A_1 \subseteq A
\end{align*}
\Rightarrow S_1 \otimes S_2 \leq_S S
\]

\[
\begin{align*}
S_1 \otimes S_2 \leq_S S \\
A_2 \supseteq A \cup A_1
\end{align*}
\Rightarrow S_2 \leq_S S/S_1
\]

\[
\begin{align*}
I_1 \models_S S_1 \text{ and } I_2 \models_S S/S_1
\end{align*}
\Rightarrow I_1 \times I_2 \models_S S
\]

\[
\forall I_1 : I_1 \models_S S_1 \Rightarrow I_1 \times I_2 \models_S S
\]

and \(A_{I_2} \supseteq A \cup A_1\)

Observe that the last sub-statement of statement 5 refines Theorem 2.4.

**Proof:**
The detailed proof of this theorem can be found in [43]. We only give here the counterexamples for the third and the fourth bullets of statement 4. First, the following counterexample shows that composition is not monotonic wrt to the weak refinement when alphabets are different. Consider the three modal specifications:

- \(S_1\) with \(A_1 = \{a\}\) and \(\text{may}_1(\epsilon) = \text{must}_1(\epsilon) = \emptyset\);
- \(S'_1\) with \(A'_1 = \{a, b\}\) and \(\text{may}'_1(\epsilon) = \{b\}\) and \(\text{must}'_1(\epsilon) = \emptyset\);
- \(S_2\) with \(A_{S_2} = \{b\}\) and \(\text{may}_2(\epsilon) = \text{must}_2(\epsilon) = \{b\}\).

Then \(S = S_1 \otimes S_2\) is defined over \(\{a, b\}\) and \(\text{may}(\epsilon) = \text{must}(\epsilon) = \{b\}\); and, \(S' = S'_1 \otimes S_2\) is defined over \(\{a, b\}\) and \(\text{may}'(\epsilon) = \{b\}\) and \(\text{must}'(\epsilon) = \emptyset\). Thus we have: \(S'_1 \leq_w S_1\) and \(S'_1 \otimes S_2 \nleq_w S_1 \otimes S_2\).

Now, this counter-example shows that \(I_1 \models_w S_1\) and \(I_2 \models_w S_2\) do not imply \(I_1 \times I_2 \models_w S_1 \otimes S_2\):

- \(S_1\) with \(A_1 = \{a\}\) and \(\text{may}_1(\epsilon) = \text{must}_1(\epsilon) = \emptyset\); \(I_1\) with \(A_{I_1} = \{a, b\}\) and \(I_1 = \{\epsilon\}\);
- \(S_2\) with \(A_2 = \{b\}\) and \(\text{may}_2(\epsilon) = \text{must}_2(\epsilon) = \{b\}\); \(I_2\) with \(A_{I_2} = \{b\}\) and \(I_2 = \{\epsilon, b\}\).

Then \(I_1 \models_w S_1\) and \(I_2 \models_w S_2\), \(I_1 \times I_2 = \{\emptyset\}\) and \(\text{may}_{S_1 \otimes S_2}(\epsilon) = \text{must}_{S_1 \otimes S_2}(\epsilon) = \{b\}\) thus \(I_1 \times I_2\) is not a weak implementation of \(S_1 \otimes S_2\). \(\square\)

**Example 2.4.** Consider now a second specification \(\text{Rel}\) for a producer in Figure 3(a) dealing with reliability: messages are negatively acknowledged until the system is reset. The specification \(\text{Fun}\) in Figure 1(a) and \(\text{Rel}\) are defined on different alphabets: the action \(\text{reset}\) is not part of the alphabet of \(\text{Fun}\) and similarly for \(\text{extra}\) in \(\text{Rel}\). The conjunction of the two aspects is depicted in Figure 3(b); observe that the modalities of the transitions labeled by \(\text{reset}\) are directly inherited from those in \(\text{Rel}\).
3. Interface automata

In [20], de Alfaro and Henzinger introduced interface automata, which 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 [20, 17] for more details.

**Definition 3.1. ([20])**

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 \( A = A? \cup A! \), 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 \( a \in A! \). We will also use \( x \xrightarrow{a} y \) to emphasize that \( (x, a, y) \in \rightarrow \). 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 semantics 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 that do not distinguish between an interface and one of its models. More precisely, the model of an interface automaton is any of its refinements. As a consequence, the notion of refinement coincides with the one of satisfaction. Moreover, any interface automaton is always satisfiable except if it is empty.

**Remark 3.1.** In interface automata, the distinction between inputs and outputs should not be interpreted as a function from the Inputs to the Outputs.

**Example 3.1.** Two interface automata are depicted in Figure 4 (this example is adapted from [20]). The client \( Cl \) in Figure 4(a) is defined over the alphabet \( \{\text{ok}?, \text{fail}?\} \cup \{\text{msg}!\} \). The action \( \text{fail}? \) never occurs which encodes the assumption that the environment of the client never transmits a \( \text{fail} \) to the client. The server \( Serv \) in Figure 4(b) is defined over the alphabet \( \{\text{msg}?, \text{ack}?, \text{nack}?, \text{sent}!, \text{ok}!, \text{fail}!\} \); when \( \text{msg} \) is invoked, the server tries to send the message and resends it if the first transmission fails. If both transmissions fail, the component reports failure (\( \text{fail}! \)), otherwise it reports success (\( \text{ok}! \)).
Alternatively, properties of interfaces are described in game-based logics, such as ATL or ATL∗[1] whose complexities are PSPACE and PTIME-complete, respectively. 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. This problem is known to be PTIME-complete. There is no notion of component reuse and shared refinement is defined in an ad-hoc manner [22].

Figure 4. Two interface automata to be composed

Remark 3.2. Contrary to input/output automata, interface automata are generally not input-enabled2. Refinement of input/output automata corresponds to simulation between traces. If the model was not input-enabled, then a refinement could accept less inputs than its abstraction. The game-based approach allows us to avoid such a situation even when the system is not input enabled.

The main advantage of the game-based approach appears in the definition of composition and compatibility between interface automata. Following [17], two interface automata are composable if they have disjoint sets of output actions, compose by synchronizing shared actions and interleave asynchronously all other actions.

Definition 3.2. (Product of interface automata [20])
Let \( P_1 = (X_1, x_{01}, A_1, \rightarrow_1) \) and \( P_2 = (X_2, x_{02}, A_2, \rightarrow_2) \) be two interface automata. The product over \( P_1 \) and \( P_2 \) is an interface automaton \( P_1 \times P_2 = (X, x_0, A, \rightarrow) \), 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 \( (x_1, y_1) \xrightarrow{a} (x_2, y_2) \) if and only 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 \( (x_1, y_1) \xrightarrow{a?} (x_2, y_2) \) if and only if there exists \( (x_1) \xrightarrow{a?_1} (x_2) \) and \( (y_1) \xrightarrow{a?_2} (y_2) \).

2Recall that a system is input-enabled if it can react to any input action in any moment.
For each $a \in (A_1 \cap A_2) \cup (A_2 \cap A_1)$, there exists a transition $(x_1, y_1) \overset{a_1}{\rightarrow} (x_2, y_2)$ iff there exists $(x_1) \overset{a}{\rightarrow} 1 (x_2)$ and $(y_1) \overset{a}{\rightarrow} 2 (y_2)$.

Since interface automata are not necessarily input-enabled (which is what allows the automaton to make assumptions on the environment), the product $P_1 \times P_2$ of two interface automata $P_1$ and $P_2$ may have 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 semantics. Indeed, the game semantics supports 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. This can be viewed as a reachability game whose complexity is linear [20]. 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.

Let $\text{Illegal}(P_1, P_2)$ be the subset of pairs $(x_1, x_2) \in X_1 \times X_2$ such that there exists

\[
\text{either an action } a \in A_1 \cap A_2 \text{ with } x_1 \overset{a}{\rightarrow}_1 \text{ but not } x_2 \overset{a}{\rightarrow}_2 \\
\text{or an action } a \in A_2 \cap A_1 \text{ with } x_2 \overset{a}{\rightarrow}_2 \text{ but not } x_1 \overset{a}{\rightarrow}_1
\]

where $x \overset{a}{\rightarrow}$ means that $x \overset{a}{\rightarrow} y$ for some state $y$. If illegal states exist in the product $P_1 \times P_2$, there may still exist refinements of the product without illegal state. Those refinements specify how the resulting product should be restricted in order to guarantee that illegal states cannot be reached. As proved in [20], there is one such largest refinement which can be 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 $z$ such that $z \overset{a}{\rightarrow}_i y$ for some $y \in Y$ and $a! \in A!$ (an output action of the product). Let $\text{pre}_0(Y) = Y$ and, for $k \geq 0$,
\[ \text{pre}_i^{k+1}(Y) = \text{pre}_i(\text{pre}_i^k(Y)) \quad \text{and let} \quad \text{pre}_i^*(Y) = \bigcup_k \text{pre}_i^k(Y). \]

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 \( P_1 \parallel P_2 \) and is called the \textit{composition} of the two interface automata. \( P_1 \) and \( P_2 \) are called \textit{compatible} if applying the pruning leaves the initial state [20]. We now recall the two following theorems from [20] that show that interface automata support independent design and substitutability.

**Theorem 3.1.** ([20])

The composition operation for interface automata is associative and commutative.

**Theorem 3.2.** ([20])

Let \( P_1, P_2, \) and \( P_3 \) be three interface automata. If \( P_2 \) refines \( P_1 \) and the set of shared actions of \( P_2 \parallel P_3 \) is included in the set of shared actions of \( P_1 \parallel P_3 \), then \( P_2 \parallel P_3 \) refines \( P_1 \parallel P_3 \).

**Example 3.2.** The product of the interface automata in Figure 4 is represented in Figure 5(a). The gray state is illegal as the server wants to report a failure (\texttt{fail!}) which is not accepted as an input by the client.

The result of applying the pruning operation is then depicted in Figure 5(b).

Bhaduri has proposed a quotient operation that is the adjoint of the composition operation [10]. This quotient, which is defined for the deterministic fragment only, is characterized in the following theorem. Let \( P \perp \) be the interface \( P \) where input and output actions have been exchanged.

**Theorem 3.3.** ([20])

Consider two deterministic interface automata \( P_1 \) and \( P_2 \). If \( P_1 \) and \( P_2 \perp \) are compatible, then there exists \( P \) such that

1. \( P_1 \parallel P \leq P_2 \),

2. for each \( P' \) such that \( P_1 \parallel P' \leq P_2 \), we have \( P' \leq P \) and,

3. \( P \) is given by \( (P_1 \parallel P_2 \perp)^\perp \).

The theorem above states that, contrary to the case of modal automata, the quotient for interface automata can be derived from the composition operation with a simple switch operation between input and output actions.

**Remark 3.3.** The operations between interface automata that have been defined so far do not require an explicit treatment of dissimilar alphabets as is the case for modal specifications. Indeed, it is implicitly handled with the help of the game-based approach. Conjunction is not defined for interface automata. For such an operation, we conjecture that the game-based approach is not powerful enough for an implicit treatment.
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 us to define 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 [31, 36]. 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 [31, 36] 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 modalities.

4.1. Profiles

Given an alphabet of actions $A$, a profile $\pi$ is a function $\pi : A \mapsto \{?, !\}$, labeling actions with the symbols $?$ (for inputs) or $!$ (for outputs). We write “$a?$” (respectively, $a!$) to express that “$\pi(a) = ?$” (respectively, $\pi(a) = !$). The set of $a \in A$ such that $\pi(a) = ?$ (respectively, $\pi(a) = !$) is denoted $A?$ (respectively $A!$). We shall sometimes write by abuse of notation, $\pi = (A?, A!)$.

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 $A_2$.

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

Proposition 4.1. The refinement $\leq$ between profiles is transitive.

Product between profiles. The product between $\pi_1$ and $\pi_2$, denoted $\pi_1 \otimes \pi_2$ is defined if and only if $A_1! \cap A_2! = \emptyset$, and is equal to the profile $\pi = (A?, A!)$ over $A_1 \cup A_2$ such that:

$$\pi = \pi_1 \otimes \pi_2 : \begin{cases} A! &= (A_1! \cup A_2!) \\ A? &= (A_1? \cup A_2?) \setminus A! \end{cases}$$

Proposition 4.2. Let $\pi'_1$, $\pi'_2$, $\pi_1$, and $\pi_2$ be profiles. The product between profiles is monotonic with respect to refinement: if $\pi'_1 \leq \pi_1$ and $\pi'_2 \leq \pi_2$ and $\pi_1 \otimes \pi_2$ and $\pi'_1 \otimes \pi'_2$ are defined, then $\pi'_1 \otimes \pi'_2 \leq \pi_1 \otimes \pi_2$.

Conjunction between profiles. The conjunction of $\pi_1$ and $\pi_2$, 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 if and only if 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 symbol in $A_1$ and with $\pi_2$ for every symbol in $A_2$.

Proposition 4.3. Let $\pi_1$, $\pi_2$ and $\pi$ be profiles. Then, $\pi \leq \pi_1 \land \pi_2$ if and only if $\pi \leq \pi_1$ and $\pi \leq \pi_2$. 

Quotient between profiles. The quotient of $\pi_1$ and $\pi_2$, denoted $\pi_1 / \pi_2$, 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 when $A_1 \cap A_2 = \emptyset$, and is thus equal to the profile $\pi = (A^?, A^!)$ such that

$$\pi_1 / \pi_2 = \begin{cases} A^! = A_1^! \setminus (A_1^! \cap A_2^!) \\ A^? = [(A_1^? \cup A_2^?) \setminus A!] \cup A_2! \end{cases}$$

**Proposition 4.4.** Let $\pi$, $\pi_1$ and $\pi_2$ be profiles defined over the alphabet $A^?$ and $A^!$ respectively:

- if $\pi_1 \otimes \pi_2 \leq \pi$ and $A_2 \supseteq A \cup A_1$, then $\pi_2 \leq \pi / \pi_1$;
- if $\pi_2 \leq \pi / \pi_1$ and $A_1 \subseteq A$, then $\pi_1 \otimes \pi_2 \leq \pi$.

4.2. The framework of modal interfaces

We now formally introduce modal interfaces that are modal specifications whose actions are also labeled with input and output attributes. We will consider the language representation in the spirit of [40, 39, 42], while Larsen et al. followed the automata-based representation (the two representations are equivalent).

**Definition 4.1. (Modal interface)**

A modal interface is a pair $C = (S, \pi)$, where $S$ is a modal specification over the alphabet $A^?$ and $A^!$ and $\pi : A^S \rightarrow \{?, !\}$ is a profile.

A model for a modal interface is a pair $(I, \pi')$, where $I$ is a prefix-closed language and $\pi'$ is a profile for $I$. We say that $(I, \pi')$ strongly implements $(S, \pi)$, written $(I, \pi') \models_s (S, \pi)$, if $I \models_s S$ and $\pi' \leq \pi$, and similarly for weak implementation. We say that $(S_2, \pi_2) \leq_s (S_1, \pi_1)$ if $S_2 \leq_s S_1$ and $\pi_2 \leq \pi_1$, and analogously 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 4.2.** Consider two modal interfaces $C_1 = (S_1, \pi_1)$ and $C_2 = (S_2, \pi_2)$, and let $\star \in \{\land, \otimes, /\}$. If $\pi_1 \star \pi_2$ is defined, then

$$C_1 \star C_2 = (S_1 \star S_2, \pi_1 \star \pi_2).$$

The following theorem states that all the characteristic properties of modal specifications directly extend to modal interfaces.

**Theorem 4.1.** Propositions stated in Theorem 2.5 extend to modal interfaces.

We now recap the translation from interface automata to modal interfaces, which will help us make the link between modalities and input or output actions.
4.4. From interface automata to modal interfaces

We recap the translation from interface automata to modal automata that has been proposed in [31]. In this section, we extend this translation to modal specifications, the language-algebraic 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 $A_{S_P} = A$ and modalities are defined for all $u \in A_{S_P}^*$:

$$a? \in \text{must}_{S_P}(u) \quad \text{if} \quad u.a? \in L_P$$
$$a! \in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) \quad \text{if} \quad u.a! \in L_P$$
$$a? \in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) \quad \text{if} \quad u \in L_P$$
$$a! \not\in \text{may}_{S_P}(u) \quad \text{if} \quad u \in L_P$$
$$a \in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) \quad \text{if} \quad u \not\in L_P.$$  \hspace{1cm} (8)

Theorem 1 of [31] shows that, with the above correspondence, alternating simulation for interface automata and modal refinement for modal interfaces coincide. Regarding supports, we have:

$$L_{S_P} = L_P \cup \{u.a?.v \mid u \in L_P, u.a? \not\in L_P, v \in A_{S_P}^*\}. \hspace{1cm} (9)$$

It is worth making some comments about this translation, given by formulas (8,9). Regarding formula (9), the supporting language $L_{S_P}$ 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, $L_{S_P}$ states no particular exception handling since everything is possible. Specifying exception handling then amounts to refining this modal interface.

Formula (8) refines (9) 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 by the environment of the assumptions made by the component are seen as an exception, and that exception handling is unspecified and not mandatory. This embedding is illustrated in Figure 6 for the case of the Client of Figure 4(a).

![Figure 6. Embedding of the interface automaton C! from Figure 4(a) into a modal interface](image-url)
4.5. On compatibility for modal interfaces

In this section, we take advantage of profiles to define a notion of composition for modal interfaces including the compatibility issue introduced for interface automata. We shall recap the solution proposed in [31, 36], then we shall show a counter example to Theorem 10 in [31] and then propose our correction.

4.5.1. The composition and the bug in Theorem 10 of [31]

We now consider the notion of compatibility of 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 \( C_1 \otimes C_2 \), such that one interface may produce an output that may not be accepted as an input by the other interface:

\[
\begin{align*}
\text{either} & \quad \text{an action } a \in A_1! \cap A_2? \\
& \quad \text{with } a \in \text{may}_{1}(u_1) \setminus \text{must}_{2}(u_2) \\
\text{or} & \quad \text{an action } a \in A_2! \cap A_1? \\
& \quad \text{with } a \in \text{may}_{2}(u_2) \setminus \text{must}_{1}(u_1),
\end{align*}
\]

where \( u_1 = \text{pr}_{A_1}(u) \) and similarly \( u_2 = \text{pr}_{A_2}(u) \). In order to get rid of illegal runs, we must first consider the words \( v \) having a suffix \( v' \) such that \( v, v' \) is illegal and \( v' \) is a sequence of outputs; this way, no environment can prevent \( v' \) to occur from \( v \). For \( U \) a set of words of modal interface \( C \), let \( \text{pre}_1(U) \) be the set

\[
\text{pre}_1(U) = \{ v \in L_C \mid \exists a! \in \text{may}(v), v.a! \in U \}
\]

Let \( \text{pre}^0_1(U) = U \), and, for \( k \geq 0 \), \( \text{pre}_{1}^{k+1}(U) = \text{pre}_1(\text{pre}_{1}^{k}(U)) \). Finally, let \( \text{pre}_1^+(U) = \bigcup_k \text{pre}_1^k(U) \). The composition of two modal interfaces is obtained from their product by removing words in \( \text{pre}_1^+(U) \), 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 [31, 36] states 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\(^3\) shows that Theorem 10 in [31, 36] is wrong.

Example 4.1. Figure 7 depicts two modal interfaces \( C_1 \) and \( C_2 \); \( I_1 \) and \( I_2 \) are implementations of \( C_1 \) and \( C_2 \), respectively. Alphabets are indicated for each modal interface. Parallel composition according to [31] is named \([C_1 \parallel C_2]_0 \). Word \( c? .a! \) is illegal since in the state reached after this run: \( C_1 \) may offer \( b! \) whereas \( C_2 \) may (in fact will) not accept it. However, \( c? .a! \) is in the product of the two implementations.

\(^3\)This example is due to discussions of the authors with Barbara Jobstmann and Laurent Doyen.
4.5.2. The correction

Call exception any 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 4.3. (compatibility)
The exception language of modal interfaces $C_1$ and $C_2$ is the language $E_{C_1 \parallel C_2} = \text{pre}^*_I (\text{Illegal}(C_1, C_2))$. 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 4.4. (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 $L_{C_1 \otimes C_2}$ do
  for all $a$ in $A$ do
    if $v \not\in E_{C_1 \parallel C_2}$ and $v.a \in E_{C_1 \parallel C_2}$ then
      for all $w$ in $A^*$ do
        must$(v.a.w) := \emptyset$
        may$(v.a.w) := A$
      end for
    end if
  end for
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 $\bot$. 

Figure 7. Counterexample regarding compatibility. Grey-shaded states are to be removed.
If the environment performs an action \( a \) 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 4.2.** We now show that our relaxation allows us to correct the counter example stated in Figure 7. 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 [31, 36], where associativity is only mentioned, we can now state the following theorem:

**Theorem 4.2.** The parallel composition operator is commutative and associative.

**Proof:**

Commutativity of \( \parallel \) immediately holds by definition. We now consider associativity. Let three modal interfaces \( \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3 \). We characterize the set of illegal words in \( ([\mathcal{C}_1 \parallel \mathcal{C}_2] \parallel \mathcal{C}_3) \) and then prove that rearranging the parentheses will not change this set.

In the sequel we shall write \( A_i \), \( \text{must}_i \), \( \text{may}_i \) the elements of \( \mathcal{C}_i \) (with \( i = 1, 2, 3 \)) and \( A_{ij} \), \( \text{must}_{ij} \), \( \text{may}_{ij} \) the elements of \( \mathcal{C}_i \otimes \mathcal{C}_j \) (for \( (i, j) \in \{1, 2, 3\} \times \{1, 2, 3\} \), such that \( i \neq j \)). We shall also write \( u_i \) for \( \text{pr}_{A_i}(u) \) and \( u_{ij} \) for \( \text{pr}_{A_i \cup A_j}(u) \).

Observe first that, by definition, \( \otimes \) is associative. Moreover, a word \( u \) is illegal in \( \mathcal{C}_1 \otimes \mathcal{C}_2 \) iff

\[
\left( \text{may}_1^1(u_1) \setminus \text{must}_2^2(u_2) \right) \cup \left( \text{may}_2^1(u_2) \setminus \text{must}_1^2(u_1) \right) \neq \emptyset \tag{11}
\]

where \( u_i = \text{pr}_{A_i}(u) \) and \( \text{may}_1^1(u_1) = \text{may}_1^1(u_1) \cap A_1 \) and similarly for other cases. Then, in building \( \mathcal{C}_1 \parallel \mathcal{C}_2 \) from \( \mathcal{C}_1 \otimes \mathcal{C}_2 \), relaxation of Def. 4.4 applies to every word \( v \in \mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2} \) such that

\[
\exists b? \in \text{may}_{12}(v) : v.b? \in \mathcal{I} \tag{12}
\]

Consequently, every word in \( \mathcal{L}_{\mathcal{C}_1 \parallel \mathcal{C}_2} \):

either belongs itself to \( \mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2} \), or has a strict prefix \( v \in \mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2} \) satisfying (12).

Observe that (12) rewrites as

\[
\exists b? \in \text{may}_{12}(v), \exists w \in (A_1! \cup A_2!)^* \Rightarrow v.b?_?w \text{ satisfies } (11). \tag{14}
\]

Apply this to the pair \( (\mathcal{C}_1 \parallel \mathcal{C}_2, \mathcal{C}_3) \): word \( u \) is illegal in \( (\mathcal{C}_1 \parallel \mathcal{C}_2) \otimes \mathcal{C}_3 \) iff

\[
\left( \text{may}_{12}^1(u_{12}) \setminus \text{must}_3^2(u_3) \right) \cup \left( \text{may}_3^1(u_3) \setminus \text{must}_{12}^2(u_{12}) \right) \neq \emptyset \tag{15}
\]

where \( A_{12} \) is the alphabet of \( \mathcal{C}_1 \parallel \mathcal{C}_2 \). Let \( U \) be the set of all such \( u \)’s, and set \( \mathcal{I} = \text{pre}^*_U(U) \). Then, in building \( (\mathcal{C}_1 \parallel \mathcal{C}_2) \parallel \mathcal{C}_3 \) from \( (\mathcal{C}_1 \parallel \mathcal{C}_2) \otimes \mathcal{C}_3 \), relaxation applies to every word \( v \in \mathcal{L}_{(\mathcal{C}_1 \parallel \mathcal{C}_2) \parallel \mathcal{C}_3} \) satisfying

Lemma 4.1. Consider three modal interfaces that the refined modal interface does not introduce new shared actions.

Proof:

Let us further analyse (15). Two cases must be considered:

1. $u_{12}$ has reached a universal state of $C_1 \parallel C_2$; in this case, by (13), $u_{12}$ has a strict prefix $\hat{u}_{12} \in \mathcal{L}_{C_1 \otimes C_2}$ satisfying (12), meaning that $\hat{u}_{12} \cdot b^?$ may for some $b^?$, by subsequently performing only output actions, reach a deadlock in the product of the pair $(C_1, C_2)$.

2. $u_{12}$ has not reached a universal state of $C_1 \parallel C_2$; in this case, $u_{12} \in \mathcal{L}_{C_1 \otimes C_2}$ and

$$\begin{align*}
\text{may}_{12}(u_{12}) &= \text{may}_{1}(u_1) \cup \text{may}_{2}(u_2) \\
\text{must}_{12}(u_{12}) &= (\text{must}_{1}(u_1) \cap \text{must}_{2}(u_2)) \cup (A_1 \setminus A_2) \cup (A_2 \setminus A_1)
\end{align*}$$

Hence the non-emptiness of (15) is equivalent to $u_{12}$ causing a deadlock in the pair $(C_1, C_3)$ or the pair $(C_2, C_3)$.

Let us summarize how the two conditions (12) were rewritten:

$$\exists b^? \in \text{may}(v), \exists w \in (A_1 \cup A_2 \cup A_3)^* \Rightarrow u = v \cdot b^? \cdot w$$

satisfies the following condition:

- There exists a pair $(i, j) \in \{1, 2, 3\} \times \{1, 2, 3\}$, such that $i \neq j$ and $u_{ij} = \text{pr}_{A_i}(u)$ possesses a prefix $\hat{u}_{ij} \cdot b^?$ that may, for some $b^?$ and by subsequently performing only output actions, reach a deadlock in the pair $(C_i, C_j)$.

The bottom line is that the condition and (12) is indeed symmetric with respect to the considered three modal interfaces. This proves the associativity of $\parallel$.

As for interface automata (Theorem 4 in [20]), strong refinement preserves compatibility, assuming that the refined modal interface does not introduce new shared actions.

Lemma 4.1. Consider three modal interfaces $C_i$, $i = 1 \ldots 3$, such that $C_2 \subseteq C_1$ and $A_2 \cap A_3 \subseteq A_1 \cap A_3$.

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

Proof:

Consider an illegal word $u \in \text{Illegal}(C_2, C_3)$ for $C_2 \otimes C_3$. 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 $a \in \text{may}_2(\text{pr}_{A_2}(u))$ 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 $a \notin \text{must}_2(\text{pr}_{A_2}(u))$ and $a \in \text{may}_3(\text{pr}_{A_3}(u))$.

By Definition 2.5, $u' = \text{pr}_{A_1 \cup A_3}(u)$ belongs to $\mathcal{L}_{C_1 \otimes C_3}$. 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 2.4, either $a$ is an output of $C_1$ and an input of $C_3$, such that $a \in \text{may}_1(\text{pr}_{A_1}(u'))$ 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 $a \not\in \text{must}_1(\text{pr}_{A_1}(u'))$ 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 point of the lemma.

Consider now the second point. Recall that $A_1 \cup A_3$ is included in $A_2 \cup A_3$. Hence, the set of actions $\text{pr}_{A_1 \cup A_3}(\text{pre}^*(\text{Illegal}(C_2, C_3)))$ is included in $\text{pre}^*(\text{pr}_{A_1 \cup A_3}(\text{Illegal}(C_2, C_3)))$, which is in turn included in $\text{pre}^*(\text{Illegal}(C_1, C_3))$, thanks to the first point of the lemma. □

**Corollary 4.1. (compatibility preservation)**

Given any three modal interfaces $C_i$, $i = 1 \ldots 3$, such that $C_2 \leq_s C_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$. $C_1$ compatible with $C_3$ implies that $C_2$ and $C_3$ are also compatible.

**Proof:**

This is an immediate consequence of the previous Lemma 4.1. Assume $C_2$ and $C_3$ incompatible, meaning that $\epsilon \in \mathcal{E}_{C_2} \parallel C_3$. By Lemma 4.1, $\epsilon = \text{pr}_{A_1 \cup A_3}(\epsilon) \in \mathcal{E}_{C_1} \parallel C_3$. Hence $C_1$ and $C_3$ are also incompatible. □

Contrary to interface automata for which $C_1 \parallel C_2$ is a refinement of $C_1 \otimes C_2$ [20], relaxation of modal interfaces amounts to computing an abstraction of the product:

**Lemma 4.2.** Given two modal interfaces $C_1$ and $C_2$:

$$C_1 \otimes C_2 \leq C_1 \parallel C_2$$

**Proof:**

Two cases are possible:

- if $u \in \mathcal{E}_{C_1} \otimes C_2 \setminus \mathcal{E}_{C_1} \parallel C_2$ then $\text{must}_{C_1 \otimes C_2}(u) = \text{must}_{C_1 \parallel C_2}(u)$ and $\text{may}_{C_1 \otimes C_2}(u) = \text{may}_{C_1 \parallel C_2}(u)$;
- if $u \in \mathcal{E}_{C_1} \parallel C_2$ then $u \in \mathcal{E}_{C_1 \parallel C_2}$ and $\text{must}_{C_1 \parallel C_2}(u) = \emptyset$ and $\text{may}_{C_1 \parallel C_2}(u) = A$.

Thus, $\text{must}_{C_1 \otimes C_2}(u) \supseteq \text{must}_{C_1 \parallel C_2}(u)$ and $\text{may}_{C_1 \otimes C_2}(u) \subseteq \text{may}_{C_1 \parallel C_2}(u)$. □

Theorem 10 stated in [31, 36] now holds for the parallel composition operator.

**Theorem 4.3. (independent implementability)**

For any two modal interfaces $C_1$, $C_2$ and two implementations $(I_1, \pi_1)$, $(I_2, \pi_2)$ such that $(I_1, \pi_1) \models_s C_1$ and $(I_2, \pi_2) \models_s C_2$, it holds that $(I_1, \pi_1) \times (I_2, \pi_2) \models_s C_1 \parallel C_2$.

**Proof:**

If $(I_1, \pi_1) \models_s C_1$ and $(I_2, \pi_2) \models_s C_2$, then, by Theorem 4.1, $(I_1, \pi_1) \times (I_2, \pi_2) \models_s C_1 \otimes C_2$. By Lemma 4.2 and by the generalization of Theorem 1 in Theorem 4.1: $(I_1, \pi_1) \times (I_2, \pi_2) \models_s C_1 \parallel C_2$. □

### 5. Methodological considerations

While the framework we propose adds significant flexibility to design flows, it also raises some methodological issues that we discuss now.
As previously remarked, a designer may want to specify the aspects of the system via different interfaces (called viewpoints). In this situation, she may wonder whether the same system would be obtained by implementing all the viewpoints in a single component, or alternatively, as several components where each of them implementing some of the viewpoints. This question amounts relating the operation of product/composition to the one of conjunction. We have the following result.

**Theorem 5.1.** Let \( C_1, C_2, C_3 \) be three modal interfaces. Then,

1. \( C_1 \otimes (C_2 \land C_3) \leq (C_1 \otimes C_2) \land (C_1 \otimes C_3) \);
2. \( C_1 \parallel (C_2 \land C_3) \leq (C_1 \parallel C_2) \land (C_1 \parallel C_3) \);
3. The reverse refinements in points 1 and 2 do not hold.

**Proof:**
Recall that \( \text{may}_i \), \( \text{must}_i \) and \( A_i \) denote the elements of \( \mathcal{C}_i \) and let \( A = A_1 \cup A_2 \cup A_3 \).

**Proof of statement 1.** By definition of the GLB of modal specifications and by Theorem 2.1, we have: \( \rho(C_2 \land C_3) \leq C_2 \land C_3 \). Note that the definition of composition for modal specification can be immediately extended to *pseudo*-modal specifications with preservation of Proposition 2.3. As a result:

\[
C_1 \otimes \rho(C_2 \land C_3) \leq C_1 \otimes (C_2 \land C_3)
\]

We then can easily prove that \( \otimes \) distributes over \&. Thus:

\[
C_1 \otimes \rho(C_2 \& C_3) \leq (C_1 \otimes C_2) \& (C_1 \otimes C_3)
\]

Recall that \( \rho(S) \) is the largest modal specification (for refinement order) such that \( \rho(S) \leq S \). Thus:

\[
C_1 \otimes \rho(C_2 \& C_3) \leq \rho(C_1 \otimes C_2) \& (C_1 \otimes C_3).
\]

That is: \( C_1 \otimes (C_2 \land C_3) \leq (C_1 \otimes C_2) \land (C_1 \otimes C_3) \).

**Proof of statement 2.** Let \( u \in \mathcal{L}_{C_1 \parallel (C_2 \land C_3)} \) then:

- either \( u \in \mathcal{L}_{C_1 \otimes (C_2 \land C_3)} \);
- or, \( u \) has a strict prefix \( v \in \mathcal{L}_{C_1 \otimes (C_2 \land C_3)} \) such that \( v.b? \in \text{Illegal}(C_1, C_2 \land C_3) \) for some action \( b? \).

In the first case, according to the point 1 of Theorem 5.1, every \( a \in \text{may}_{C_1 \otimes (C_2 \land C_3)}(u) \) also belongs to \( \text{may}_{C_1 \otimes C_2 \land (C_1 \otimes C_3)}(u) \). By definition of the conjunction:

\[
a \in [\text{may}_{C_1 \otimes C_2}(\text{pr}_{A_1 \cup A_2}(u)) \cup (A \setminus (A_1 \cup A_2))] \cap [\text{may}_{C_1 \otimes C_3}(\text{pr}_{A_1 \cup A_3}(u)) \cup (A \setminus (A_1 \cup A_3))]
\]

Moreover, by Lemma 4.2, \( C_1 \otimes C_i \leq C_i \parallel C_i \) for \( i = 1, 2 \), thus:

\[
a \in [\text{may}_{C_1 \parallel C_2}(\text{pr}_{A_1 \cup A_2}(u)) \cup (A \setminus (A_1 \cup A_2))] \cap [\text{may}_{C_1 \parallel C_3}(\text{pr}_{A_1 \cup A_3}(u)) \cup (A \setminus (A_1 \cup A_3))]
\]

that is, \( a \in \text{may}_{C_1 \parallel (C_2 \parallel C_3)}(u) \).
In the second case, let us first show that if \( v.b? \in \text{Illegal}(C_1, C_2 \land C_3) \) then \( \text{pr}_{A_1 \cup A_1}(v.b?) \in \text{Illegal}(C_1, C_i) \) for \( i = 1, 2 \). Let \( \text{may}_i^1 = \text{may}_i \cap A_i \) and similarly for other cases. When \( v.b? \in \text{Illegal}(C_1, C_2 \land C_3) \) there exists \( a \in A_1 \cap A_2 \cap A_3 \) such that:

\[
a \in (\text{may}_1^1(\text{pr}_{A_1}(v.b?))) \setminus \text{must}^2_{C_2 \land C_3}(\text{pr}_{A_2 \cup A_3}(v.b?)) \\
\cup (\text{may}_1^1(\text{pr}_{A_1}(v.b?))) \setminus \text{must}_2^1(u_1(\text{pr}_{A_1}(v.b?)))
\]

If \( a \notin \text{must}^2_{C_2 \land C_3}(\text{pr}_{A_2 \cup A_3}(v.b?)) \) then \( a \notin [\text{must}^2_{C_2}(\text{pr}_{A_2}(v.b?)) \cup \text{must}^2_{C_3}(\text{pr}_{A_3}(v.b?))]; \) moreover if \( a \in \text{may}^1_{C_2 \land C_3}(\text{pr}_{A_2 \cup A_3}(v.b?)) \), as \( a \in A_1 \cap A_2 \cap A_3 \), we have: \( a \in \text{may}^1_{C_2}(\text{pr}_{A_2}(v.b?)) \cap \text{may}^1_{C_3}(\text{pr}_{A_3}(v.b?)) \). As a result, \( \text{pr}_{A_1 \cup A_1}(v.b?) \) is illegal in \( C_1 \otimes C_i \) for \( i = 1, 2 \) and \( u \) has reached a universal state in \( (C_1 \parallel C_2) \land (C_1 \parallel C_3) \). In conclusion, \( \text{may}_{C_1 \otimes (C_2 \land C_3)}(u) \subseteq \text{may}_{(C_1 \parallel C_2) \land (C_1 \parallel C_3)}(u) \).

Now if \( a \in \text{must}_{(C_1 \parallel C_2) \land (C_1 \parallel C_3)}(u) \) then by definition:

\[
a \in [(\text{must}_{1}(\text{pr}_{A_1}(u)) \cup (A_2 \setminus A_1)) \cap (\text{must}_{2}(\text{pr}_{A_2}(u)) \cup (A_1 \setminus A_2))]
\]

\[
\cup [(\text{must}_{1}(\text{pr}_{A_1}(u)) \cup (A_3 \setminus A_1)) \cap (\text{must}_{3}(\text{pr}_{A_3}(u)) \cup (A_1 \setminus A_3))]
\]

We have to prove \( a \in \text{must}_{C_1 \parallel (C_2 \land C_3)}(u) \), that is:

\[
a \in [(\text{must}_{1}(\text{pr}_{A_1}(u)) \cup ((A_2 \cup A_3) \setminus A_1))
\]

\[
\cap [(\text{must}_{2}(\text{pr}_{A_2}(u)) \cup \text{must}_{3}(\text{pr}_{A_3}(u))) \cup (A_1 \setminus (A_2 \cup A_3))]
\]

If \( a \notin \text{must}_{1}(\text{pr}_{A_1}(u)) \) then from Equation 17 we deduce:

\[
a \in \text{must}_{2}(\text{pr}_{A_2}(u)) \cap \text{must}_{3}(\text{pr}_{A_3}(u)) \cap (A_2 \setminus A_1) \cap (A_3 \setminus A_1)
\]

If \( a \in \text{must}_{1}(\text{pr}_{A_1}(u)) \) then from Equation 17 we deduce:

\[
a \in ((\text{must}_{2}(\text{pr}_{A_2}(u)) \cup (A_1 \setminus A_2)) \cap ((\text{must}_{3}(\text{pr}_{A_3}(u)) \cup (A_1 \setminus A_3)))
\]

In the two situations, Equation 18 is true and thus \( a \in \text{must}_{C_1 \parallel (C_2 \land C_3)}(u) \).

**Proof of statement 3 and 4.** Consider the three following modal interfaces defined over the alphabet \{a\} with the same profile \( \pi(a) = \vdash \):

- \( C_1 \) with \( \text{may}_1(\varepsilon) = \{a\} \), \( \text{may}_1(a) = \{a\} \) and \( \text{may}_1(aa) = \emptyset \);
- \( C_2 \) with \( \text{may}_2(\varepsilon) = \{a\} \), \( \text{may}_2(a) = \{a\} = \emptyset \);
- \( C_3 \) with \( \text{may}_3(\varepsilon) = \{a\} \), \( \text{may}_3(a) = \text{must}_3(a) = \{a\} \) and \( \text{may}_1(aa) = \emptyset \);

Then \( \text{may}_{(C_1 \otimes C_2) \land (C_1 \otimes C_3)}(\varepsilon) = \{a\} \) whereas \( \text{may}_{C_1 \otimes (C_2 \land C_3)}(\varepsilon) = \emptyset \). As a result:

\[
(C_1 \otimes C_2) \land (C_1 \otimes C_3) \notin (C_1 \otimes (C_2 \land C_3)).
\]

The same counterexample can be used to prove that: \( (C_1 \parallel C_2) \land (C_1 \parallel C_3) \notin (C_1 \parallel (C_2 \land C_3)) \). □
The interpretation of this theorem is as follows. We assume two components indexed by 1 and 2, with associated interfaces. The left hand side of equations in point 1 and 2 captures the design process in which (1) the two viewpoints for component 2 are first combined, and (2) the two components are combined; this is called a \textit{component-centric} design process because it aims at specifying components completely before assembling them. The right hand side captures the design process in which viewpoints are first considered separately for all components, and then fused; this is called a \textit{viewpoint-centric} design process. Theorem 5.1 expresses that viewpoint-centric design processes leave more room for implementations than component-centric ones.

6. Conclusion, related work and future work

This paper presents a \textit{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 [42]). However, the core contributions of the paper are (1) a parallel composition operator that reflects a rich notion of compatibility between components, actually correcting the parallel composition proposed in [31, 36], and (2) a new theory that encompasses dissimilar alphabets.

Interface automata were first introduced as an extension of Input/Output automata with an optimistic approach for composition. Modal specifications have been proposed as an extension of process-algebraic theories [35, 30] which allows for a better distinction between successive implementations (see the introduction of [36] and [30] for some discussion). Modal interfaces are a model that mixes both I/O automata and modal specifications.

There are various other approaches for interface theories (see [5] for a survey). One of them is based on contracts [7, 36, 38, 27], that is a representation where one keeps an explicit distinction between assumptions on the environment and guarantees on behaviors of the system. A similar approach to ours has been developed in [33] for a \textit{non-modal} process-algebraic framework in which a dedicated predicate is used to model inconsistent processes.

Interface automata and modal specifications are incomparable models as \textit{must}, \textit{may} and \textit{input}, \textit{output} have orthogonal meanings. Modal specification can be viewed as an abstraction of a set of closed systems\footnote{A closed system is a system that does not interact with an unknown environment. On the contrary, an open system is a system that continuously interacts with an unknown environment.} (as a modal specification does not allow a component and its environment to be distinguished). As a consequence, specification logics and verification procedures for this model [26, 28] are extensions of those defined for transition systems [37, 15]. Interface automata is a more “open” model (as it distinguishes between the component and its environment) and it is thus not surprising that specification logics and verification procedures for such a model correspond to those defined for reactive systems, e.g., ATL [1]. This paper did not focus on verification procedures, but we believe that this research direction is of importance and deserves further studies.

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 [13, 18]. One should also propose definitions of quotient and conjunction for interface automata.
Another promising direction would be a timed extension of modal interfaces. In [21], de Alfaro et al. proposed timed interface automata that extend timed automata just as interface automata extend finite-word automata. The semantics of a timed interface automaton is given by a timed game [19, 12], which allows one to capture the timed dimension in composition, i.e., “what are the temporal ordering constraints on communication events between components? [21]”. Up to now, composition is the only operation that has been defined on timed interface automata. In [14], Chatain et al. have proposed a notion of refinement for timed games. However, monotonicity of parallel composition with respect to this refinement relation has not been investigated yet. In [9, 8], 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.

Acknowledgments

We are grateful to Barbara Jobstmann and Laurent Doyen who proposed the counter example given in Section 4.5.1 which proved that the construction in [31] was incorrect.

References


