2009 Ninth International Conference on Application of Concurrency to System Design

Why are modalities good for Interface Theories?

Jean-Baptiste Raclet*, Eric Badouel†, Albert Benveniste†, Benoît Caillaud† and Roberto Passerone‡
*INRIA, Grenoble, France. Email: raclet@inrialpes.fr
†INRIA/IRISA, Rennes, France. Email: surname.name@inria.fr
‡PARADES Scarl, Rome, Italy and University of Trento, Trento, Italy. Email: roberto.passerone@unitn.it

Abstract

In this paper we revisit the fundamentals of interface theories. Methodological considerations call for supporting "aspects" and "assume/guarantee" reasoning. From these considerations, we show that, in addition to the now classical refinement and substitutability properties of interfaces, two additional operations are needed, namely: conjunction *and* residuation *(or* quotient*). We draw the attention to the difficulty in handling interfaces having different alphabets — which calls for alphabet equalization. We show that alphabet equalization must be performed differently for the different operations. Then, we show that* Modal Interfaces*, as adapted from the original proposal by Kim Larsen, offer the needed flexibility.*

1. Introduction

Context: Interfaces have emerged as an essential concept for component-based system engineering. According to our understanding of industrial needs, a theory of interfaces is subject to the following list of requirements:

1) *Locality of alphabets*: Large systems are composed of many subsystems possessing their own alphabet of ports and variables. Handling different alphabets for different subsystems or components may seem like a trivial requirement but has not been properly addressed by some theories.
2) *Substitutability*: Subsystems or components should be designable in isolation, by including the needed information regarding possible future contexts of use. When developed independently, subsystems or components should be substitutable to their specifications and compose as expected.
3) *Contracts*: Complex embedded and reactive systems are generally developed under a multi-layered OEM-supplier chain. Hence, interface theories should offer provision for contractual relations by formalizing, for a considered subsystem, a contract consisting of: 1) its context of use (*assumptions*), and 2) what is expected from the subsystem (*guarantee*).
4) *Multiple aspects or viewpoints*: Large systems are concurrently developed for its 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 which is central in Time-Triggered development disciplines [1], and *memory* and *power* aspects. 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.
5) *Conjunctive requirements*: It is the current practice that early requirements capture relies on Doors sheets, or even Excel files containing a bench of textual requirements, with little formal support. 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 [2], whose sentences are translatable into predefined behavioral patterns. Alternatively, graphical scenario languages could be considered [3], [4]. In any case, many such requirements would remain attached to a given (sub)system. This requires being able to support the concept of *conjunction* of requirements in our interface theory.

Interfaces have been the subject of considerable literature, see [5] for an in-depth bibliographical study.

In 2001, de Alfaro and Henzinger [6] introduced *Interface Automata*, where interfaces are seen as games between the component and its environment. Since then, Interface automata have often been considered as *the* theory of reference regarding interfaces. Refinement is by *alternating simulation* [7], which amounts to getting more permissive regarding the environment and more constrained regarding the considered component. Parallel composition is monotonous with respect to refinement and ensures substitutability and deadlock freeness. This framework was adapted in [8] to synchronous symbolic transition systems and was subsequently extended to handling shared refinement [9]. How-

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

ever, requirements 4 on multiple aspects and 5 on conjunctive requirements, and even the obvious requirement 1 fail to be properly addressed in the above theories.

An extensive trace-based theory of Assume/Guarantee reasoning and contracts has been proposed in [10] with explicit handling of multiple-viewpoint contracts. Still, requirement 1 is not properly addressed, as we shall see.

Building on [6] in combination with background work on *modal automata* [11], Larsen et al. [12] have shown that the framework of Interface Automata is naturally embedded into that of *Modal I/O Automata*, a slight variation of modal automata. According to this embedding, alternating simulation appears as a particular case of modal refinement. In [13], the same group of authors adapts modal I/O automata to support Assume/Guarantee reasoning. Regarding the variations around the generic concept of modality, an extensive bibliographical study is again found in [5]. This is a fundamental step as it allows replacing the sophisticated, game oriented, refinement by alternating simulation, by the much simpler notion of *modal refinement*. Still, our requirements 1, 4, and 5 are not clearly met, although provision was available in this work to achieve this.

In his thesis [14], Raclet provided an interesting language-oriented variation of modal automata, called *modal specifications*. Modal specifications are the language version of modal automata. They correspond to the conjunctive fragment of the mu-calculus [15], [16]. They are slightly more restrictive than modal automata, because, by not handling states explicitly, they cannot capture nondeterminism. On the other hand, they are more elegant in that modal refinement is sound *and complete* for modal specifications — see [17] regarding the non-completeness of modal refinement, for modal automata.

Contribution: In this paper we further develop the approach of [12] to address our above requirements on Interface Theories. We build on the framework of modal specifications proposed by Raclet [14], [18]. Modal specifications come equipped with several operations: *composition* $\otimes$ and a *refinement* order $\leq$, which in turn induces the greatest lower bound (GLB) $\wedge$.

Our first contribution is to show that the operation of GLB allows addressing multiple-viewpoint and conjunctive requirements. Specifically, a key contribution is the clarification of the role of modalities in handling specifications with different alphabets of actions. We show that alphabet equalization must be performed *differently*, depending on whether parallel composition or conjunction is considered. Then we show that, in performing alphabet equalization, modalities offer the needed flexibility, whereas other formalisms do not.

Our second contribution concerns Assume/Guarantee reasoning. Article [13] proposes such a framework on top of I/O automata. It consists in specifying a pair (A, G) of assumption and guarantee, where A and G are two I/O automata with the constraint that $A \otimes G$ is a closed system (with empty environment). Our contribution to Assume/Guarantee reasoning consists in the formalization of contracts as quotients or *residuations* G/A, where G are the guarantees and A the assumptions both specified as modal specifications. This residuation "/" is indeed the adjoint of composition $\otimes$ and captures in an algebraic setting the intuition of implication that underpins assume/guarantee reasoning.

Compatibility and deadlock freeness are important issues raised by de Alfaro and Henzinger in [6]. They are, however, orthogonal to the above discussed ones and are therefore not addressed here.

Organization: The paper is organized as follows. Modal specifications are recalled in section 2 for the case of a fixed alphabet. In section 3 we recall the translation of Interface Automata into Modal Automata proposed in [12] and we explain why Interface Automata are not prepared to handle conjunction with different alphabets. Dealing with different alphabets is investigated in section 4 for the framework of Modal Specifications. In section 5 we further discuss why Modal Specification properly address our requirements for a theory of interfaces. And, finally, we conclude.

2. Modal Specifications for the case of a fixed alphabet

Our background material is borrowed from [12], [18] and we mostly use notations from the latter reference. The notion of modal specification proposed in [18] is just a language-oriented rephrasing of the concept of modal automaton of [12]. In this section we assume a *fixed* alphabet A of actions.

Definition 1 (modal specification) *A* modal specification *is a tuple $\mathcal{S} = (A, must, may)$, where*

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

are partial functions satisfying the following consistency *condition:*

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

The intended meaning is that, for $u \in A^*$, $a \in may(u)$ means that action a is allowed after u, $a \in must(u)$ means that action a is required after u, $a \notin may(u)$ means that action a is disallowed after u, often written $a \in mustnot(u)$. We shall sometimes write $A_\mathcal{S}$, $may_\mathcal{S}$, and $must_\mathcal{S}$ to refer to the entities involved in the definition of $\mathcal{S}$.

A triple $\mathcal{S}$ satisfying definition 1 with the exception of (1) is called a *pseudo-modal specification.* For ${}^p\mathcal{S}$ a pseudo-modal specification, a word $u \in A^*$ is called *consistently specified* in ${}^p\mathcal{S}$ if it satisfies (1); ${}^p\mathcal{S}$ itself is called *consistent* if every $u \in A^*$ is consistently specified in it; i.e., ${}^p\mathcal{S}$ is a modal specification if and only if it is consistent. For ${}^p\mathcal{S} = (A, must, may)$ a pseudo-modal specification, the *support* of ${}^p\mathcal{S}$ is the least language $\mathcal{L}_{{}^p\mathcal{S}}$ such that:

(i) $\epsilon \in \mathcal{L}_{{}^p\mathcal{S}}$, where ϵ denotes the empty word; and
(ii) $u \in \mathcal{L}_{{}^p\mathcal{S}}$ and $a \in may(u)$ imply $u.a \in \mathcal{L}_{{}^p\mathcal{S}}$.

Definition 2 (implementation) *A prefix-closed language $\mathcal{I} \subseteq A^*$ is an* implementation *of pseudo-modal specification ${}^p\mathcal{S} = (A, must, may)$, denoted by $\mathcal{I} \models {}^p\mathcal{S}$, if:*

$$\forall u \in \mathcal{I} \quad \Rightarrow \quad must(u) \subseteq \mathcal{I}_u \subseteq may(u)$$

where $\mathcal{I}_u$ is the set of actions $a \in A$ such that $u.a \in \mathcal{I}$.

Lemma 1 *If $\mathcal{I} \models {}^p\mathcal{S}$, then $\mathcal{I} \subseteq \mathcal{L}_{{}^p\mathcal{S}}$ holds and every word of $\mathcal{I}$ is consistently specified in ${}^p\mathcal{S}$.*

The concept of *thorough refinement* [17] follows immediately from definition 2 by comparing, through set inclusion, the sets of implementations associated to two modal specifications. Thorough refinement has been extensively studied in [17] and compared to the more syntactic notion of *modal refinement* that we recall next. We will use modal refinement in this article.

Definition 3 (modal refinement) *Say that ${}^p\mathcal{S}_1$* refines *${}^p\mathcal{S}_2$, written ${}^p\mathcal{S}_1 \leq {}^p\mathcal{S}_2$, iff for all $u \in \mathcal{L}_{{}^p\mathcal{S}_1}$, $may_{{}^p\mathcal{S}_1}(u) \subseteq may_{{}^p\mathcal{S}_2}(u)$ and $must_{{}^p\mathcal{S}_1}(u) \supseteq must_{{}^p\mathcal{S}_2}(u)$.*

Refinement is a preorder relation. However it implies inclusion of supports: $\mathcal{L}_{{}^p\mathcal{S}_1} \subseteq \mathcal{L}_{{}^p\mathcal{S}_2}$. Any two modal specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \leq \mathcal{S}_2 \leq \mathcal{S}_1$ have equal supports $\mathcal{L} = \mathcal{L}_{\mathcal{S}_1} = \mathcal{L}_{\mathcal{S}_2}$ and for all $u \in \mathcal{L}$, $may_{\mathcal{S}_1}(u) = may_{\mathcal{S}_2}(u)$ and $must_{\mathcal{S}_1}(u) = must_{\mathcal{S}_2}(u)$. Said differently, equivalent modal specifications differ only outside of their support. A unique representant $\mathcal{S} = (A, must, may)$ of equivalence classes of modal specifications is defined by assuming that for all $u \notin \mathcal{L}_\mathcal{S}$, $must(u) = \emptyset$ and $may(u) = A$. In the sequel, only modal specifications satisfying this property are considered. Under this assumption, modal refinement is a partial order relation on modal specifications.

Moreover, it is shown in [14], [18] that modal refinement for modal specifications is sound *and complete,* i.e., is equivalent to thorough refinement[1]. The following result relates implementations to consistency, for a pseudo-modal specification:

Theorem 1 (consistency [14], [18]) *Either pseudo-modal specification ${}^p\mathcal{S}$ possesses no implementation, or there exists a largest (for refinement order) modal specification $\rho({}^p\mathcal{S})$ having the same alphabet of actions and such that $\rho({}^p\mathcal{S}) \leq {}^p\mathcal{S}$. In addition, $\rho({}^p\mathcal{S})$ possesses the same set of implementations as ${}^p\mathcal{S}$. Modal specification $\rho({}^p\mathcal{S})$ is called the* pruning *of ${}^p\mathcal{S}$.*

The modal specification $\rho({}^p\mathcal{S})$ is obtained from ${}^p\mathcal{S}$ through the following steps:

1) Start from R_0, a copy of ${}^p\mathcal{S}$;
2) Let U_0 be the set of words u inconsistently specified in R_0, meaning that u does not satisfy condition (1). For each $u \in U_0$, set $may_{R_0}(u) = A$ and $must_{R_0}(u) = \emptyset$. 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 $may_{R_0}(v)$. Performing these two operations yields a pseudo-modal specification R_1 such that U_0 is consistently specified in R_1. Since we only have removed inconsistently specified words from $\mathcal{L}_{R_0}$, by Lemma 1, R_1 and R_0 possess identical sets of implementations.
3) Observe that, if, however, $a \in must_{R_1}(v)$, 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) $may_{R_{k+1}}(v) \subseteq may_{R_k}(v)$ 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.
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 ${}^p\mathcal{S}$ is removed after finitely many steps of the above algorithm. This proves that the procedure eventually converges. The limit $\rho({}^p\mathcal{S})$ is consistent and is given by:

$$may(u) = \bigcap_k may_{R_k}(u)$$
$$must(u) = \begin{cases} must_{{}^p\mathcal{S}}(u) & \text{if } must_{{}^p\mathcal{S}}(u) \subseteq may(u) \\ \emptyset & \text{otherwise} \end{cases}$$

The above procedure terminates in finitely many steps if the pseudo-modal specification is rational, i.e., originates from a *deterministic pseudo-modal automaton* [14], [18].

Greatest Lower Bound: addressing requirements 4 and 5. The set of all pseudo-modal specifications equipped with modal refinement $\leq$ is a lattice. We denote by ${}^p\mathcal{S}_1 \& {}^p\mathcal{S}_2$ the *Greatest Lower Bound* (GLB) of ${}^p\mathcal{S}_1$ and ${}^p\mathcal{S}_2$. The GLB ${}^p\mathcal{S} = {}^p\mathcal{S}_1 \& {}^p\mathcal{S}_2$ is defined by:

$$\begin{array}{rcl} may_{{}^p\mathcal{S}}(u) & = & may_{{}^p\mathcal{S}_1}(u) \cap may_{{}^p\mathcal{S}_2}(u) \\ must_{{}^p\mathcal{S}}(u) & = & must_{{}^p\mathcal{S}_1}(u) \cup must_{{}^p\mathcal{S}_2}(u) \end{array} \quad (2)$$

1. Completeness of modal refinement does not hold for nondeterministic modal automata [17]. It holds in our case since we work with specifications (for which determinism is hardwired), not automata.

Observe that, even if ${}^p\mathcal{S}_1$ and ${}^p\mathcal{S}_2$ satisfy (1), it is not guaranteed that ${}^p\mathcal{S}_1 \& {}^p\mathcal{S}_2$ does too. Hence, by using theorem 1, for $\mathcal{S}_1$ and $\mathcal{S}_2$ two modal specifications, we define $\mathcal{S}_1 \wedge \mathcal{S}_2$ as being the (uniquely defined) modal specification

$$\mathcal{S}_1 \wedge \mathcal{S}_2 = \rho(\mathcal{S}_1 \& \mathcal{S}_2). \tag{3}$$

GLB satisfies the following key property, which relates GLB to logic formulas, cf. requirements 4 and 5:

Theorem 2 (conjunctive interfaces [14], [18])

$$\mathcal{I} \models \mathcal{S}_1 \wedge \mathcal{S}_2 \;\Leftrightarrow\; \mathcal{I} \models \mathcal{S}_1 \textit{ and } \mathcal{I} \models \mathcal{S}_2$$

The following holds regarding supports: $\mathcal{L}_{\mathcal{S}_1 \wedge \mathcal{S}_2} \subseteq \mathcal{L}_{\mathcal{S}_1} \cap \mathcal{L}_{\mathcal{S}_2}$*, with equality if and only if no pruning is needed, i.e.,* $\mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{S}_1 \& \mathcal{S}_2$.

Let $\mathcal{I} \subseteq A^*$ be a prefix-closed language. It can be seen as the modal specification $\mathcal{S}_\mathcal{I}$ which admits $\mathcal{I}$ as unique implementation. It is defined as follows: $\mathcal{S}_\mathcal{I} = (A, must, may)$, with $\forall u \in A^*, must(u) = may(u) = \mathcal{I}_u$. Using this embedding of prefix-closed languages in modal specifications, the following result refines theorem 1. It uses the *least upper bound* (LUB) of modal specifications $\mathcal{S}_1 \vee \mathcal{S}_2$, obtained by taking the union of *may* and intersection of *must* — observe that, unlike for GLB, no risk of inconsistency can occur.

Lemma 2 *For* ${}^p\mathcal{S}$ *a pseudo-modal specification, its pruning* $\rho({}^p\mathcal{S})$*, as defined in theorem 1, satisfies* $\rho({}^p\mathcal{S}) = \bigvee_{\mathcal{I} \models {}^p\mathcal{S}} \mathcal{S}_\mathcal{I}$.

Composition: addressing requirement 2. For $\mathcal{S}_1$ and $\mathcal{S}_2$ two modal specifications, their *composition* $\mathcal{S} = \mathcal{S}_1 \otimes \mathcal{S}_2$ is defined by

$$\begin{array}{rcl} may_\mathcal{S}(u) & = & may_{\mathcal{S}_1}(u) \;\cap\; may_{\mathcal{S}_2}(u) \\ must_\mathcal{S}(u) & = & must_{\mathcal{S}_1}(u) \;\cap\; must_{\mathcal{S}_2}(u) \end{array} \tag{4}$$

Note that consistency raises no difficulty here. Composition ensures substitutability, cf. requirement 2:

Theorem 3 (substitutability in composition [14], [18])

1) *If* $\mathcal{S}'_1 \leq \mathcal{S}_1$ *and* $\mathcal{S}'_2 \leq \mathcal{S}_2$*, then* $\mathcal{S}'_1 \otimes \mathcal{S}'_2 \leq \mathcal{S}_1 \otimes \mathcal{S}_2$.
2) *If* $\mathcal{I}_1 \models \mathcal{S}_1$ *and* $\mathcal{I}_2 \models \mathcal{S}_2$*, then* $\mathcal{I}_1 \times \mathcal{I}_2 \models \mathcal{S}_1 \otimes \mathcal{S}_2$*, where* $\mathcal{I}_1 \times \mathcal{I}_2 = \mathcal{I}_1 \cap \mathcal{I}_2$.
3) *The following holds regarding supports:* $\mathcal{L}_{\mathcal{S}_1 \otimes \mathcal{S}_2} = \mathcal{L}_{\mathcal{S}_1} \cap \mathcal{L}_{\mathcal{S}_2}$.

Residuation: addressing requirement 3. As said before, we will also make use of the operation of *residuation*, introduced by Raclet [14], [18], which we will show (theorem 4) to be the adjoint of composition. For $\mathcal{S}_1$ and $\mathcal{S}_2$ two modal specifications, we first define their *pseudo-quotient* ${}^p\mathcal{S} = \mathcal{S}_1 /\!\!/ \mathcal{S}_2$ according to the following disjunctive and exhaustive cases:

$$\begin{array}{lll} a \in may_{{}^p\mathcal{S}}(u) \cap must_{{}^p\mathcal{S}}(u) & \text{if} & a \in must_{\mathcal{S}_1}(u) \\ & \text{and} & a \in must_{\mathcal{S}_2}(u) \\ a \in must_{{}^p\mathcal{S}}(u) \setminus may_{{}^p\mathcal{S}}(u) & \text{if} & a \in must_{\mathcal{S}_1}(u) \\ & \text{and} & a \notin must_{\mathcal{S}_2}(u) \\ a \in may_{{}^p\mathcal{S}}(u) \setminus must_{{}^p\mathcal{S}}(u) & \text{if} & a \in may_{\mathcal{S}_1}(u) \\ & \text{and} & a \notin must_{\mathcal{S}_1}(u) \\ a \in may_{{}^p\mathcal{S}}(u) \setminus must_{{}^p\mathcal{S}}(u) & \text{if} & a \notin may_{\mathcal{S}_1}(u) \\ & \text{and} & a \notin may_{\mathcal{S}_2}(u) \\ a \notin may_{{}^p\mathcal{S}}(u) \cup must_{{}^p\mathcal{S}}(u) & \text{if} & a \notin may_{\mathcal{S}_1}(u) \\ & \text{and} & a \in may_{\mathcal{S}_2}(u) \end{array}$$

Observe that, due to the second case, $\mathcal{S}_1 /\!\!/ \mathcal{S}_2$ is not consistent. Having defined $\mathcal{S}_1 /\!\!/ \mathcal{S}_2$, using the pruning operation of theorem 1, we can now set

$$\mathcal{S}_1 / \mathcal{S}_2 = \rho(\mathcal{S}_1 /\!\!/ \mathcal{S}_2) \tag{5}$$

Observe that, even if $\mathcal{S}_1$ and $\mathcal{S}_2$ are two prefix-closed languages, i.e., $\forall u, must_{\mathcal{S}_i}(u) = may_{\mathcal{S}_i}(u)$ for $i = 1, 2$, quotient $\mathcal{S}_1/\mathcal{S}_2$ is nevertheless a modal specification that is not a language.

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

Theorem 4 (residuation and contracts [14], [18])

1) $\mathcal{S}_1 \otimes \mathcal{S}_2 \leq \mathcal{S}$ *if and only if* $\mathcal{S}_2 \leq \mathcal{S}/\mathcal{S}_1$
2) $\forall \mathcal{I}_1 : [\mathcal{I}_1 \models \mathcal{S}_1 \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models \mathcal{S}]$ *iff* $\mathcal{I}_2 \models \mathcal{S}/\mathcal{S}_1$.

By theorem 4, residuation properly addresses requirement 3 regarding contracts: if $\mathcal{A}$ and $\mathcal{G}$ are modal specifications representing assumptions and guarantees, then $\mathcal{C} = \mathcal{G}/\mathcal{A}$ adequately represents the contract assumptions $\Rightarrow$ guarantees. Indeed, if environment $\mathcal{I}_E$ realizes $\mathcal{A}$ and $\mathcal{I}$ realizes $\mathcal{C}$, then $\mathcal{I} \times \mathcal{I}_E$ ($\mathcal{I}$ put in the context of environment $\mathcal{I}_E$) realizes $\mathcal{G}$.

Discussion: So far this collects all operations we need in the case of a fixed alphabet. To deal with different alphabets, the standard approach consists in first equalizing alphabets of different specifications, and then applying the above defined operations. Thus a careful study of alphabet equalization is needed. Prior to addressing the case of different alphabets, we shall first recall the mapping of Interface Automata to Modal I/O Automata as reported in [12]. This will allow us to explain why there is a fundamental problem with Interface Automata in dealing with different alphabets in the context of conjunction.

3. Mapping Interface Automata to Modal Specifications: a difficulty

An *Interface Automaton* [6] 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. Split A into $A? \uplus A! =$ input $\uplus$ output actions. We do not consider internal actions and consider only deterministic transition relations. Symbols $a?$, $a!$ and a denote elements of $A?$, $A!$ and A, respectively. Write $x \xrightarrow{a} y$ to mean $(x, a, y) \in \rightarrow$.

There are two central aspects in the theory of Interface Automata: alternating simulation, which defines refinement, and compatibility, which addresses deadlock-freeness. In this paper, we consider only the first issue and leave the second one for another work.

For $\mathcal{P}_1$ and $\mathcal{P}_2$ two Interface Automata, a binary relation $\Delta \subseteq X_1 \times X_2$ is called an *alternating simulation* of $\mathcal{P}_1$ by $\mathcal{P}_2$ if:

1. $(x_{0,1}, x_{0,2}) \in \Delta$
2. $\left.\begin{array}{c} (x_1, x_2) \in \Delta \\ x_2 \xrightarrow{a?} y_2 \end{array}\right\} \Rightarrow \exists y_1 \in X_1 : \left\{\begin{array}{l} x_1 \xrightarrow{a?} y_1 \\ (y_1, y_2) \in \Delta \end{array}\right.$
3. $\left.\begin{array}{c} (x_1, x_2) \in \Delta \\ x_1 \xrightarrow{a!} y_1 \end{array}\right\} \Rightarrow \exists y_2 \in X_2 : \left\{\begin{array}{l} x_2 \xrightarrow{a!} y_2 \\ (y_1, y_2) \in \Delta \end{array}\right.$

and Δ is the largest relation satisfying the above conditions. Say that $\mathcal{P}_2$ *simulates* or *refines* $\mathcal{P}_1$ if such a Δ exists. There is no notion of implementation for Interface Automata. Nevertheless, we may, for convenience, agree that $\mathcal{P}_2$ *implements* $\mathcal{P}_1$ if $\mathcal{P}_2$ refines $\mathcal{P}_1$.

The embedding of Interface Automata [6] into Modal I/O automata proposed in [12] extends, mutatis mutandis, to Modal Specifications, except that we must restrict ourselves to considering only deterministic Interface Automata. For completeness, we recall here this embedding. The translation function $\mathcal{P} \mapsto \mathcal{S}_\mathcal{P}$ is given next, where $\mathcal{L}_\mathcal{P}$ denotes the (prefix-closed) language defined by $\mathcal{P}$. The alphabet of $\mathcal{S}_\mathcal{P}$ is $A_{\mathcal{S}_\mathcal{P}} = A_\mathcal{P}$ and modalities are defined for all $u \in A^*_\mathcal{P}$:

$$\begin{array}{lll} a? \in must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a? \in \mathcal{L}_\mathcal{P} \\ a! \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a! \in \mathcal{L}_\mathcal{P} \\ a? \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\ & \text{and} & u.a? \notin \mathcal{L}_\mathcal{P} \\ a! \notin may_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\ & \text{and} & u.a! \notin \mathcal{L}_\mathcal{P} \\ a \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \notin \mathcal{L}_\mathcal{P} \end{array} \quad (6)$$

Theorem 6 of [12] shows that, with the above correspondence, alternating simulation and modal refinement coincide, for interface automata on the one hand, and for modal interfaces on the other hand. Regarding supports, we have:

$$\mathcal{L}_{\mathcal{S}_\mathcal{P}} = \mathcal{L}_\mathcal{P} \uplus \{u.a?.v \mid u \in \mathcal{L}_\mathcal{P}, u.a? \notin \mathcal{L}_\mathcal{P}, v \in A^*_\mathcal{P}\} \quad (7)$$

It is worth making some comments about this translation, given by formulas (6,7). Regarding formula (7), the supporting language $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ allows the environment to violate the constraints set on it by the interface automaton $\mathcal{P}$. When this happens — formally, the environment exits the alternating simulation relation — the component considers that the assumptions under which it was supposed to perform are violated, so it allows itself breaching its own promises and can perform anything afterwards. One could also see the violation of assumptions as an exception. Then, $\mathcal{L}_{\mathcal{S}_\mathcal{P}}$ states no particular exception handling since everything is possible. Specifying exception handling then amounts to refining this modal interface.

Formula (6) refines (7) 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 or local 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.

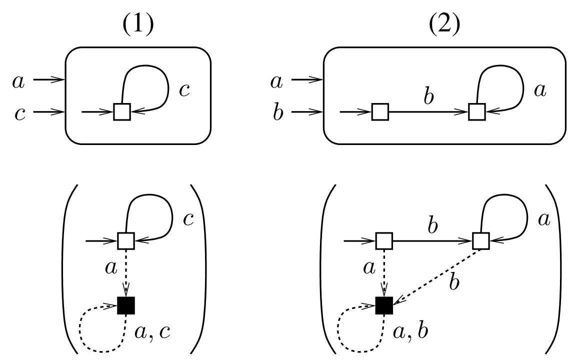

Figure 1. Translation of two Interface Automata (top) into corresponding Modal Specifications (bottom).

This translation is illustrated in figure 1. In this and the following figures, $may \setminus must$ and $must$ transitions are depicted using dashed and solid arrows, respectively. The input/output status of each action is indicated on the interface profile of the two Interface Automata. For instance, the first Interface automaton has alphabet $\{a?, c?\}$ (it has no output), and input $a?$ is not within the assumed actions from the environment. The resulting Modal Specifications are input-enabled (i.e., from every state, for every $a?$, there exists an outgoing may transition labeled by $a?$) . However, when the environment violates the assumptions, then a transition to the "black" state occurs, where any subsequent behavior can occur. Black states capture exceptions.

Why are Interface Automata not well prepared to encompass conjunction? To discuss this, let us informally reformulate the two Interface Automata of figure 1 as the following sentences:

(1) Environment shall not perform a; it may perform c repeatedly;
(2) Environment shall first perform b; then it may perform a repeatedly.

Specification (1) does not speak about b. To prepare for conjunction with specification (2), we may want to extend specification (1) to b as well. Let us try this on specification (1) in its Interface Automaton form, directly. The two

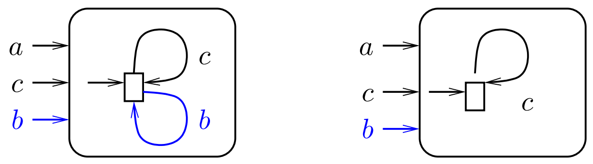

Figure 2. Extending Interface automaton (1) to b, two guesses.

possible guesses are shown on figure 2. None of them is satisfactory. The first one says that the environment is *allowed* to perform b; unfortunately this may contradict another specification that would assume that b is never performed by the environment. The second one says that the environment *shall not* perform b; unfortunately this indeed contradicts specification (2). Indeed, none of the two extensions is *neutral* with regard to b, they rather all say something about it.

So the reader may wonder whether we should not simply *change* the way we extend Interface Automata to larger alphabets. Changing is not the right answer, however, as the first solution to extend specification (1) in figure 2 is the right one in a context of parallel composition. In fact, *different* alphabet extensions are needed for to deal with parallel composition and conjunction. As we shall see now, modalities appear as an elegant solution to address alphabet equalization with appropriate flexibility.

4. Dealing with different alphabets

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 $\mathbf{pr}_B(w)$ 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 $\mathcal{L}_{\downarrow B} = \{u \in B^* \mid u = \mathbf{pr}_B(w), w \in \mathcal{L}\}$ and the *extension* of $\mathcal{L}$ to C is the language $\mathcal{L}_{\uparrow C} = \{u \in C^* \mid \mathbf{pr}_A(u) \in \mathcal{L}\}$. 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

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

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

We now introduce the different alphabet extensions we need in our theory of Modal Specifications. This is a key contribution of our work as it will provide us with a very elegant way of dealing with different alphabets.

Definition 4 (weak and strong extensions) *Let* ${}^{p}\mathcal{S} = (A, must_{{}^p\mathcal{S}}, may_{{}^p\mathcal{S}})$ *be a pseudo-modal specification and let* $C \supseteq A$*.*

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

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

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

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

Regarding supports, the following equalities hold: $\mathcal{L}_{(\mathcal{S}_{\Uparrow C})} = \mathcal{L}_{(\mathcal{S}_{\uparrow C})} = (\mathcal{L}_{\mathcal{S}})_{\uparrow C}$. We are now ready to extend the operations of section 2 to the general case.

Definition 5 *In the following,* ${}^{p}\mathcal{S}, {}^{p}\mathcal{S}_i$ *and* $\mathcal{S}_i$ *denote pseudo-modal or modal specifications over alphabets* $A_{{}^p\mathcal{S}}, A_{{}^p\mathcal{S}_i}, A_{\mathcal{S}_i}$*, for* $i = 1, 2$*, respectively. The relations and operations of section 2 are redefined as follows:*

$$\begin{aligned} &[\textit{weak implementation; } C \supseteq A_{{}^p\mathcal{S}}] \\ &\qquad \mathcal{I} \subseteq C^* \models_w {}^{p}\mathcal{S} \quad \textit{iff} \quad \mathcal{I} \models {}^{p}\mathcal{S}_{\Uparrow C} \\ &[\textit{strong implementation; } C \supseteq A_{{}^p\mathcal{S}}] \\ &\qquad \mathcal{I} \subseteq C^* \models_s {}^{p}\mathcal{S} \quad \textit{iff} \quad \mathcal{I} \models {}^{p}\mathcal{S}_{\uparrow C} \end{aligned}$$

$$\begin{aligned} &[\textit{weak refinement; } A_{\mathcal{S}_2} \supseteq A_{\mathcal{S}_1}] \\ &\qquad {}^{p}\mathcal{S}_2 \leq_w {}^{p}\mathcal{S}_1 \quad \textit{iff} \quad {}^{p}\mathcal{S}_2 \leq {}^{p}\mathcal{S}_{1\Uparrow A_{\mathcal{S}_2}} \\ &[\textit{strong refinement; } A_{\mathcal{S}_2} \supseteq A_{\mathcal{S}_1}] \\ &\qquad {}^{p}\mathcal{S}_2 \leq_s {}^{p}\mathcal{S}_1 \quad \textit{iff} \quad {}^{p}\mathcal{S}_2 \leq {}^{p}\mathcal{S}_{1\uparrow A_{\mathcal{S}_2}} \end{aligned}$$

$$\begin{aligned} &[\textit{operators; } A = A_{\mathcal{S}_1} \cup A_{\mathcal{S}_2}] \\ &\qquad \mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{S}_{1\Uparrow A} \wedge \mathcal{S}_{2\Uparrow A} \\ &\qquad \mathcal{S}_1 \otimes \mathcal{S}_2 = \mathcal{S}_{1\uparrow A} \otimes \mathcal{S}_{2\uparrow A} \\ &\qquad \mathcal{S}_1 \,/\, \mathcal{S}_2 = \mathcal{S}_{1\Uparrow A} \,/\, \mathcal{S}_{2\uparrow A} \end{aligned}$$

Note the careful use of weak and strong extensions in the different operations. The results of section 2 are slightly weakened as indicated next.

Theorem 5 (See [19] for a proof.)

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

$$\models_s \subseteq \models_w \quad \textit{and} \quad \leq_s \subseteq \leq_w$$

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

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

3) *The following holds regarding conjunction:*

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

4) *Theorem 3 regarding composition 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.*

5) *Theorem 4 is modified as follows:*

$$\left.\begin{array}{r} \mathcal{S}_2 \leq_s \mathcal{S}/\mathcal{S}_1 \\ A_{\mathcal{S}_1} \subseteq A_{\mathcal{S}} \end{array}\right\} \Rightarrow \mathcal{S}_1 \otimes \mathcal{S}_2 \leq_s \mathcal{S}$$

$$\left.\begin{array}{r} \mathcal{S}_1 \otimes \mathcal{S}_2 \leq_s \mathcal{S} \\ A_{\mathcal{S}_2} \supseteq A_{\mathcal{S}} \cup A_{\mathcal{S}_1} \end{array}\right\} \Rightarrow \mathcal{S}_2 \leq_s \mathcal{S}/\mathcal{S}_1$$

$$\left.\begin{array}{r} \mathcal{I}_1 \models_s \mathcal{S}_1 \text{ and } \mathcal{I}_2 \models_s \mathcal{S}/\mathcal{S}_1 \\ A_{\mathcal{S}_1} \subseteq A_{\mathcal{S}} \end{array}\right\} \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S}$$

$$\left.\begin{array}{c} \forall \mathcal{I}_1 : \mathcal{I}_1 \models_s \mathcal{S}_1 \\ \Downarrow \\ \mathcal{I}_1 \times \mathcal{I}_2 \models_s \mathcal{S} \\ \text{and } A_{\mathcal{I}_2} \supseteq A_{\mathcal{S}} \cup A_{\mathcal{S}_1} \end{array}\right\} \Rightarrow \mathcal{I}_2 \models_s \mathcal{S}/\mathcal{S}_1$$

Regarding statement 2, recall that modal refinement is not complete w.r.t. thorough refinement for *nondeterministic* modal automata, as shown by Nyman et al. in [17], [5], even for a fixed alphabet. Also, observe that the last sub-statement of statement 5 refines theorem 4.

5. Discussion: why are Modal Specifications appropriate?

In this section we further discuss the relative merits of Modal specifications in handling our requirements for an interface theory.

The conjunction with different alphabets, back to the example of figures 1 and 2. The conjunction of the two modal specifications of figure 1 is shown in figure 3. The self-loops attached to the large parentheses indicate the effect of weak extension: the indicated self-loops distribute over all states of the specification sitting inside the corresponding parentheses. In the conjunction, a pair of black states yields a black state, and a pair of white/black states yields a shaded state.

Observe that, in contrast to figure 2 for Interface Automata, the extension performed on the first Modal Specification (shown on top-left) is really neutral with regard to b, because of the following rule that immediately follows from (2):

$$\begin{array}{c} a \in may_{\mathcal{S}_1}(u) \text{ and } a \in whatever_{\mathcal{S}_2}(u) \\ \Downarrow \\ a \in whatever_{\mathcal{S}_1 \wedge \mathcal{S}_2}(u) \end{array}$$

for every u that is consistently defined in $\mathcal{S}_1 \& \mathcal{S}_2$, where *whatever* denotes any one of the modalities

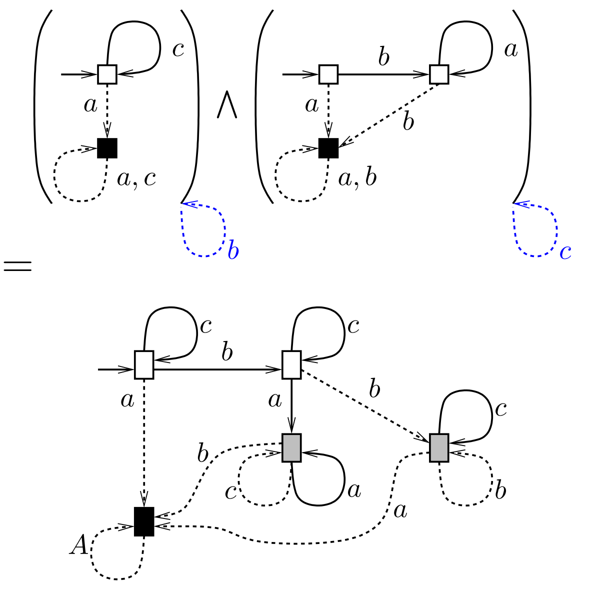

Figure 3. Conjunction of modal specifications of figure 1.

$may, must, mustnot$. Thus, modalities appear as an elegant solution to address alphabet equalization.

Assume/Guarantee reasoning. In [10], a direct, trace-theoretic approach to Assume/Guarantee reasoning is proposed. This approach builds on the concept of *contract*, which consists of a pair $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, where $\mathcal{A}$ and $\mathcal{G}$, the *assumptions* and *guarantees*, are prefix-closed languages. The alphabet of contract $\mathcal{C}$ is defined as $A_{\mathcal{C}} = A_{\mathcal{A}} \cup A_{\mathcal{G}}$. In this context, an *implementation* is a language $\mathcal{I}$ such that 1) $A_{\mathcal{I}} \supseteq A_{\mathcal{C}}$, and 2) $\mathcal{I} \times \mathcal{A} \subseteq \mathcal{G}_{\uparrow A_{\mathcal{I}}}$ where $\times$ denotes the shuffle product of languages. From this notion of implementation, a notion of refinement follows: assuming $A_2 \supseteq A_1$, $\mathcal{C}_2 \leq \mathcal{C}_1$ holds if $\mathcal{A}_2 \supseteq (\mathcal{A}_1)_{\uparrow A_2}$ and $\mathcal{G}_2 \subseteq (\mathcal{G}_1)_{\uparrow A_2}$. Greatest lower bound (representing conjunction of contracts) is then defined by

$$\mathcal{C}_1 \wedge \mathcal{C}_2 = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{G}_1 \cap \mathcal{G}_2)$$

after proper alphabet equalization by extension. Finally, a parallel composition of contracts is defined by setting, again after equalization by extension:

$$\mathcal{C}_1 \otimes \mathcal{C}_2 = ((\mathcal{A}_1 \cap \mathcal{A}_2) \cup \neg(\mathcal{G}_1 \cap \mathcal{G}_2), \mathcal{G}_1 \cap \mathcal{G}_2)$$

The same criticism applies to this approach regarding the handling of assumptions with unequal alphabets, since alphabet equalization is performed via (strong) extension.

Article [13] proposes a framework for Assume/Guarantee reasoning by building on top of I/O automata. It consists in specifying a pair $(\mathcal{A}, \mathcal{G})$ of assumption and guarantee, where A and G are two I/O automata with the constraint that $A \otimes G$ is a closed system (with empty environment). A comprehensive theory of refinement is proposed that nicely ensures substitutability. The theory is fine, but we think that the particular discipline that 1) pair $(\mathcal{A}, \mathcal{G})$ must yield a closed system, and 2) an interface is specified by only one such pair, makes this framework hardly practical as a user oriented specification formalism.

We propose the following alternative approach. First, we allow for any user-oriented formalism to specify pairs $\{\text{assumption}, \text{guarantee}\}$. Such a formalism might be textual (semi-formal natural language, translated to regular expressions), or it might be graphical, e.g., scenario languages such as LSCs [3] or HMSCs [4]. A pair $(\mathcal{A}, \mathcal{G})$ is then translated into a pair of modal specifications and the resulting contract $\mathcal{C} = \mathcal{G}/\mathcal{A}$ follows, based on theorem 4. Contracts are then handled by using our theory.

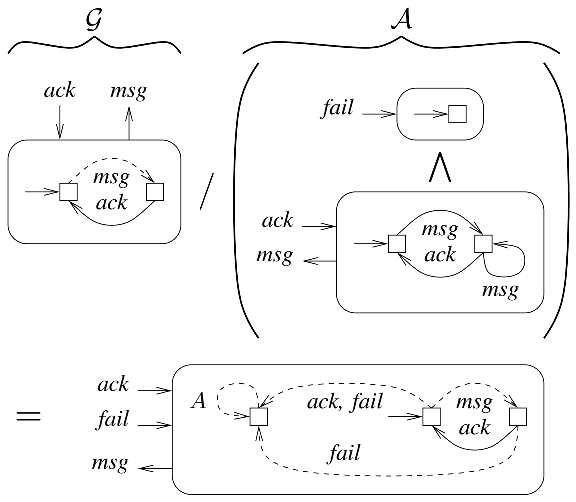

Figure 4. Representing a pair $\{\text{Assumption}, \text{Guarantee}\}$ by a contract $\mathcal{C} = \mathcal{G}/\mathcal{A}$.

Such a representation is illustrated in figure 4, showing a *send-ack* protocol seen as a service from the point of view of its user. The user is guaranteed that she may send a message and the protocol must respond by an *ack*. The protocol assumes that the underlying network does not fail (first conjunct) and never repeats an *ack*.

6. Conclusion

In this paper we have revisited some of the fundamentals of interface theories. Methodological considerations call for supporting "aspects" or and "assume/guarantee" reasoning. We have shown that, in addition to the now classical refinement and substitutability properties of interfaces, two additional operations are needed, namely: *conjunction* and *residuation*. We have highlighted the difficulty in handling interfaces having different alphabets. We have shown that alphabet equalization must be performed differently for the different operations. Then, we have shown that *Modal Interfaces*, as adapted by Raclet from the original proposal by Kim Larsen, offer the needed flexibility, whereas several formalisms fail.

Further issues include 1) the comparison of our approach with de Alfaro-Henzinger Interface Automata and the study of deadlock-freeness and compatibility, and 2) the development of a similar theory for *synchronous systems* together with a corresponding algorithmic toolbox implementing the framework and its operations, plus services such as refinement and consistency checking.

ACKNOWLEDGEMENT. Axel Legay is acknowledged for fruitful comments on an earlier version of this paper.

References

[1] H. Kopetz, "The time-triggered model of computation," in *Proceedings of the 19th IEEE Real-Time Systems Symposium, RTSS'98*, 1998, pp. 168–177.

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

[3] W. Damm and D. Harel, "Lscs: Breathing life into message sequence charts," *Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, 2001.

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

[5] U. Nyman, "Modal transition systems as the basis for interface theories and product lines," Ph.D. dissertation, Aalborg University, Department of Computer Science, September 2008.

[6] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE'01)*. ACM Press, 2001, pp. 109–120.

[7] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi, "Alternating refinement relations," in *Proceedings of the 9th International Conference on Concurrency Theory, CONCUR'98*, 1998, pp. 163–178.

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

[9] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov, "Interface theories with component reuse," in *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT'08*, October 2008, pp. 79–88.

[10] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects, FMCO'07*, ser. Lecture Notes in Computer Science, vol. 5382. Springer, October 2008, pp. 200–225.

[11] K. G. Larsen, “Modal specifications,” in *Automatic Verification Methods for Finite State Systems*, ser. Lecture Notes in Computer Science, vol. 407. Springer, 1989, pp. 232–246.

[12] K. G. Larsen, U. Nyman, and A. Wasowski, “Modal I/O automata for interface and product line theories,” in *Programming Languages and Systems, 16th European Symposium on Programming, ESOP'07*, ser. Lecture Notes in Computer Science, vol. 4421. Springer, 2007, pp. 64–79.

[13] ——, “Interface input/output automata,” in *14th International Symposium on Formal Methods, FM'06*, ser. Lecture Notes in Computer Science, vol. 4085. Springer, 2006, pp. 82–97.

[14] J.-B. Raclet, “Quotient de spécifications pour la réutilisation de composants,” Ph.D. dissertation, Ecole doctorale Matisse, université de Rennes 1, November 2007.

[15] G. Feuillade and S. Pinchinat, “Modal specifications for the control theory of discrete-event systems.” *Discrete Event Dynamic Systems*, vol. 17, no. 2, pp. 211–232, 2007.

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

[17] K. G. Larsen, U. Nyman, and A. Wasowski, “On modal refinement and consistency,” in *Proceedings of the 18th International Conference on Concurrency Theory, CONCUR'07*. Springer Verlag, 2007, pp. 105–119, lecture Notes in Computer Science.

[18] J.-B. Raclet, “Residual for component specifications,” in *Proceedings of the 4th International Workshop on Formal Aspects of Component Software, FACS'07*, Sophia-Antipolis, France, September 2007.

[19] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone, “Why are modalities good for interface theories?” INRIA, Research report, April 2009, extended version.