Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive, 9 - 38123 POVO, Trento - Italy
http://disi.unitn.it

Contract model operators for composition and merging: extensions and proofs

Roberto Passerone, Íñigo Íncer Romeo, and Alberto L. Sangiovanni-Vincentelli

August 2019

Technical Report # DISI-19-004

Contract model operators for composition and merging: extensions and proofs

ROBERTO PASSERONE, University of Trento
ÍÑIGO ÍNCER ROMEO, University of California, Berkeley
ALBERTO L. SANGIOVANNI-VINCENTELLI, University of California, Berkeley

1 INTRODUCTION

This report is an extension of the paper "Coherent Extension, Composition, and Merging Operators in Contract Models for System Design", published as part of the EMSOFT 2019 proceedings on ACM Transactions on Embedded Computing Systems [70]. We complete the work with the proofs of all results, and extend the treatment of the related work. In this report, we replicate the structure of that paper to match the definition and theorem numbers, although we do not repeat the text which can be found in the original publication.

Several component, interface and contract models have been proposed in the literature [9, 12, 13, 22, 26, 29, 33, 40, 47, 48, 56, 63, 64, 75, 76]. In this report, we examine the general form of an interface or contract, and study equivalent representations that facilitate the definition of the operators and refinement relations. We deal with refinement between components at the same level of abstraction, which we call *conformance* to distinguish from other forms of refinement. In particular, we are interested in conformance and its related conjunction operator applied across different *viewpoints*. We adapt the mathematical framework proposed by Benveniste et al. [10, 27], with simplifications to facilitate the exposition. The reader is referred to our full paper [70] for comments, explanations and examples regarding the definitions and results outlined in this report.

2 COMPONENT, INTERFACES AND CONTRACT MODELS

2.1 Components

Definition 2.1 (Component). Let $\Sigma \subseteq \mathcal{A}$ be an alphabet, and let $\mathcal{B}(\Sigma)$ be the set of all behaviors over alphabet Σ. A *component* M over alphabet Σ is a set of behaviors from $\mathcal{B}(\Sigma)$, i.e.,

$$M \subseteq \mathcal{B}(\Sigma).$$

A component may further partition its alphabet into a set of inputs I and outputs O, such that $I \cup O = \Sigma$.

Definition 2.2 (Component composition). Let M_1 and M_2 be components over alphabet Σ, and let $O_1 \cap O_2 = \emptyset$. Their composition $M = M_1 \parallel M_2$ is the component over alphabet Σ given by

$$M = M_1 \cap M_2.$$

Definition 2.3 (Component projection). Let M be a component over alphabet Σ', and let $\Sigma \subseteq \Sigma'$. The projection of M to Σ is defined as

$$\mathbf{proj}_{\Sigma',\Sigma}(M) = \{x \in \mathcal{B}(\Sigma) \mid \exists y \in M, x = \mathbf{proj}_{\Sigma',\Sigma}(y)\}.$$

Definition 2.4 (Component inverse projection). Let M be a component over alphabet Σ, and let $\Sigma' \supseteq \Sigma$. The inverse projection of M to Σ' is defined as

$$\mathbf{proj}^{-1}_{\Sigma',\Sigma}(M) = \{x \in \mathcal{B}(\Sigma') \mid \mathbf{proj}_{\Sigma',\Sigma}(x) \in M\}.$$

Authors' addresses: Roberto Passerone, roberto.passerone@unitn.it, University of Trento, Dipartimento di Ingegneria e Scienza dell'Informazione, Via Sommarive 9, Trento, Italy, 38123; Íñigo Íncer Romeo, inigo@eecs.berkeley.edu, University of California, Berkeley, Electrical Engineering and Computer Sciences, Berkeley, CA, 94720; Alberto L. Sangiovanni-Vincentelli, alberto@eecs.berkeley.edu, University of California, Berkeley, Electrical Engineering and Computer Sciences, Berkeley, CA, 94720.

Definition 2.5 (Component conformance). Let M and M' be components over the same alphabet Σ, such that $I_1 = I_2$ and $O_1 = O_2$. Then, M conforms to M', written $M \preceq M'$, whenever

$$M \subseteq M'.$$

Definition 2.6 (Component complementation). Let M be a component over alphabet Σ. The complement $\overline{M}$ of M is the component given by

$$\overline{M} = \mathcal{B}(\Sigma) - M.$$

Theorem 2.7 (Component conjunction and disjunction). *Let M_1 and M_2 be components over alphabet Σ, such that $I_1 = I_2$ and $O_1 = O_2$. The conjunction $M = M_1 \sqcap M_2$ is the component over alphabet Σ given by*

$$M = M_1 \cap M_2.$$

The disjunction $M = M_1 \sqcup M_2$ is the component over alphabet Σ given by

$$M = M_1 \cup M_2.$$

Proof. Consider the case for conjunction. We prove that M is the greatest lower bound of conformance. We must first show that M conforms to M_1 and M_2. This follows easily, since by hypothesis $M \subseteq M_1$ and $M \subseteq M_2$. Thus, by Definition 2.5, $M \preceq M_1$ and $M \preceq M_2$.

Assume now M' conforms to both M_1 and M_2. We must show that M' conforms to M. Assume this is not the case. Then, by Definition 2.5, there exists $x \in M'$ such that $x \notin M$. Because M' conforms to M_1 and M_2, x must necessarily be in at least one of these two. If $x \in M_1$, then necessarily $x \notin M_2$, otherwise, by hypothesis, $x \in M_1 \cap M_2$, and therefore $x \in M$. This contradicts the assumption that M' conforms to M_2. Similarly if $x \in M_2$. Thus, by contradiction, M' conforms to M.

The case for disjunction is similar. □

2.2 Interfaces

Definition 2.8 (Interface). An interface $\mathcal{I}$ over alphabet Σ is a pair $(\mathcal{R}, \mathcal{G})$ where $\mathcal{R}$ (the requirements) and $\mathcal{G}$ (the guarantees) are components over Σ.

Definition 2.9 (Implementations and Environments). A component M is an *implementation* of the interface $\mathcal{I} = (\mathcal{R}, \mathcal{G})$, written $M \models_G \mathcal{I}$, if and only if $M \preceq \mathcal{G}$. It is an *environment* of $\mathcal{I}$, written $E \models_R \mathcal{I}$, if and only if $E \preceq \mathcal{R}$.

Definition 2.10. Let $\mathcal{I}_1 = (\mathcal{R}_1, \mathcal{G}_1)$ and $\mathcal{I}_2 = (\mathcal{R}_2, \mathcal{G}_2)$ be interfaces. Then,

$$\mathcal{I}_1 \parallel \mathcal{I}_2 = (\mathcal{R}_1 \parallel \mathcal{R}_2, \mathcal{G}_1 \parallel \mathcal{G}_2).$$

Definition 2.11 (Interface conformance and equivalence). Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ and $\mathcal{I}' = (\mathcal{R}', \mathcal{G}')$ be interfaces. We define

- $\mathcal{I} \preceq \mathcal{I}'$ if and only if $\mathcal{R}' \preceq \mathcal{R}$ and $\mathcal{G} \preceq \mathcal{G}'$.
- $\mathcal{I} \sim \mathcal{I}'$ if and only if $\mathcal{R}' \sim \mathcal{R}$ and $\mathcal{G} \sim \mathcal{G}'$.

Corollary 2.12. *An interface $\mathcal{I}$ conforms to an interface $\mathcal{I}'$, written $\mathcal{I} \preceq \mathcal{I}'$, if and only if, for all components M and E,*

$$M \models_G \mathcal{I} \implies M \models_G \mathcal{I}' \quad \text{and} \quad E \models_R \mathcal{I}' \implies E \models_R \mathcal{I}.$$

Proof. For the forward direction, assume $\mathcal{I}$ conforms to $\mathcal{I}'$. Let now $M \models_G \mathcal{I}$. Then, by Definition 2.9, $M \preceq \mathcal{G}$. By Definition 2.11, since $\mathcal{I} \preceq \mathcal{I}'$, $\mathcal{G} \preceq \mathcal{G}'$. Thus, by transitivity, $M \preceq \mathcal{G}'$. Hence, $M \models_G \mathcal{I}'$. Similarly for an environment E.

For the reverse direction, assume for all M, $M \models_G \mathcal{I} \implies M \models_G \mathcal{I}'$. By Definition 2.9, this is the same as $M \preceq \mathcal{G} \implies M \preceq \mathcal{G}'$. We want to show that $\mathcal{G} \preceq \mathcal{G}'$. Assume this is not the case. Then, there exists $x \in \mathcal{G}$ such that $x \notin \mathcal{G}'$. Choose $M = \{x\}$. Then, $M \preceq \mathcal{G}$, however $M \not\preceq \mathcal{G}'$, which contradicts the hypothesis. Hence, $\mathcal{G} \preceq \mathcal{G}'$. Similarly, we show that $\mathcal{R}' \preceq \mathcal{R}$. Therefore, by Definition 2.11, $\mathcal{I} \preceq \mathcal{I}'$. □

Lemma 2.13 (Interface Bounds). *An interface $\mathcal{I}$ is a greatest lower bound of the interfaces $\mathcal{I}_1$ and $\mathcal{I}_2$, written $\mathcal{I} = \mathcal{I}_1 \sqcap \mathcal{I}_2$, if and only if, for all components M and E,*

$$M \models_G \mathcal{I} \Leftrightarrow M \models_G \mathcal{I}_1 \wedge M \models_G \mathcal{I}_2 \quad \textit{and}$$
$$E \models_R \mathcal{I} \Leftrightarrow E \models_R \mathcal{I}_1 \vee E \models_R \mathcal{I}_2.$$

It is a least upper bound, written $\mathcal{I} = \mathcal{I}_1 \sqcup \mathcal{I}_2$, if and only if, for all components M and E,

$$M \models_G \mathcal{I} \Leftrightarrow M \models_G \mathcal{I}_1 \vee M \models_G \mathcal{I}_2 \quad \textit{and}$$
$$E \models_R \mathcal{I} \Leftrightarrow E \models_R \mathcal{I}_1 \wedge E \models_R \mathcal{I}_2.$$

Proof. Assume $\mathcal{I} = \mathcal{I}_1 \sqcap \mathcal{I}_2$. Then, by definition, $\mathcal{I} \preceq \mathcal{I}_1$ and $\mathcal{I} \preceq \mathcal{I}_2$, and therefore, by Corollary 2.12, $M \models_G \mathcal{I} \implies M \models_G \mathcal{I}_1 \wedge M \models_G \mathcal{I}_2$. Assume now $M \models_G \mathcal{I}_1 \wedge M \models_G \mathcal{I}_2$, and assume, by contradiction, that $M \not\models_G \mathcal{I}$. Then construct the interface $\mathcal{I}'$ with $\mathcal{G}' = \mathcal{G}_1 \cap \mathcal{G}_2$ and $\mathcal{R}' = \mathcal{R}_1 \cup \mathcal{R}_2$. Clearly, $\mathcal{I}'$ is a lower bound of $\mathcal{I}_1$ and $\mathcal{I}_2$. Thus, by definition of greatest lower bound, $\mathcal{I}' \preceq \mathcal{I}$, which implies $M \models_G \mathcal{I}$, contradicting our hypothesis.

The remaining clauses can be proven similarly. □

Theorem 2.14. *Let $\mathcal{I}_1 = (\mathcal{R}_1, \mathcal{G}_1)$ and $\mathcal{I}_2 = (\mathcal{R}_2, \mathcal{G}_2)$ be interfaces. Then,*

$$\mathcal{I}_1 \sqcap \mathcal{I}_2 = (\mathcal{R}_1 \sqcup \mathcal{R}_2, \mathcal{G}_1 \sqcap \mathcal{G}_2) \quad \textit{and}$$
$$\mathcal{I}_1 \sqcup \mathcal{I}_2 = (\mathcal{R}_1 \sqcap \mathcal{R}_2, \mathcal{G}_1 \sqcup \mathcal{G}_2).$$

Proof. The theorem follows from Lemma 2.13. Construct the given interfaces, and apply Lemma 2.13 in the backward direction to show that the expressions indeed form the greatest lower bound and the least upper bound. □

2.3 Contracts

Definition 2.15 (Maximal System). The maximal system of an interface $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ is the component $S_{\mathcal{I}} = \mathcal{G} \parallel \mathcal{R}$.

Definition 2.16 (System Equivalence). Two interfaces $\mathcal{I}_1$ and $\mathcal{I}_2$ are *system equivalent*, written $\mathcal{I}_1 \equiv \mathcal{I}_2$, if and only if $S_{\mathcal{I}_1} = S_{\mathcal{I}_2}$.

Lemma 2.17. $\mathcal{I}_1 \sim \mathcal{I}_2 \implies \mathcal{I}_1 \equiv \mathcal{I}_2$.

Proof. Assume $\mathcal{I}_1 \sim \mathcal{I}_2$. Then, by Definition 2.11, $\mathcal{R}_2 \sim \mathcal{R}_1$ and $\mathcal{G}_1 \sim \mathcal{G}_2$. By Definition 2.5, conformance for components is the same as set containment. Hence, $\mathcal{R}_1 = \mathcal{R}_2$ and $\mathcal{G}_1 = \mathcal{G}_2$. Therefore, by Definition 2.15, $S_{\mathcal{I}_1} = S_{\mathcal{I}_2}$ Hence, by Definition 2.16, $\mathcal{I}_1 \equiv \mathcal{I}_2$. □

Definition 2.18 (Extension). An interface $\mathcal{I}'$ is an extension of an interface $\mathcal{I}$, written $\mathcal{I} \rightsquigarrow \mathcal{I}'$, if and only if, for all components M and E,

$$M \models_G \mathcal{I} \implies M \models_G \mathcal{I}' \quad \text{and} \quad E \models_R \mathcal{I} \implies E \models_R \mathcal{I}'.$$

Corollary 2.19. $\mathcal{I} \rightsquigarrow \mathcal{I}' \iff \mathcal{G} \subseteq \mathcal{G}' \wedge \mathcal{R} \subseteq \mathcal{R}'$.

Proof. For the forward direction, assume $\mathcal{I}'$ is an extension of $\mathcal{I}$. By Definition 2.9, $\mathcal{G} \models_G \mathcal{I}$. Therefore, by Definition 2.18, $\mathcal{G} \models_G \mathcal{I}'$. Thus, by Definition 2.9, $\mathcal{G} \preceq \mathcal{G}'$. Hence, by Definition 2.5, $\mathcal{G} \subseteq \mathcal{G}'$. The proof is similar for $\mathcal{R}$.

For the reverse direction, assume $\mathcal{G} \subseteq \mathcal{G}'$. Let $M \models_G \mathcal{I}$. Then, by Definition 2.9, $M \preceq \mathcal{G}$, and therefore, by Definition 2.5, $M \subseteq \mathcal{G}$. Thus, by transitivity, $M \subseteq \mathcal{G}'$. Therefore, by reasoning backwards, $M \models_G \mathcal{I}'$. By a similar argument, we show that $E \models_R \mathcal{I} \implies E \models_R \mathcal{I}'$. Hence, by Definition 2.18, $\mathcal{I} \rightsquigarrow \mathcal{I}'$. □

Definition 2.20 (Completion). An interface $\mathcal{I}'$ is a *completion* of an interface $\mathcal{I}$, written $\mathcal{I} \mapsto \mathcal{I}'$, if and only if $\mathcal{I} \equiv \mathcal{I}'$ and $\mathcal{I} \rightsquigarrow \mathcal{I}'$.

Definition 2.21 (Maximal Interface). An interface $\mathcal{I}$ is maximal if and only if it is maximal under completion, i.e., for all interfaces $\mathcal{I}'$, if $\mathcal{I} \mapsto \mathcal{I}'$, then $\mathcal{I} \sim \mathcal{I}'$.

Theorem 2.22. *Let $\mathcal{B}(\Sigma)$ be the universe of behaviors over alphabet Σ. An interface $\mathcal{I}$ over Σ is maximal if and only if $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$.*

Proof. For the forward direction, let $\mathcal{I}$ be a maximal interface and, by way of contradiction, assume that $\mathcal{G} \cup \mathcal{R} \neq \mathcal{B}(\Sigma)$. Then, there must be $x \in \mathcal{B}(\Sigma)$ such that $x \notin \mathcal{G}$ and $x \notin \mathcal{R}$. Construct interface $\mathcal{I}' = (\mathcal{R}', \mathcal{G}')$, such that $\mathcal{G}' = \mathcal{G} \cup \{x\}$ and $\mathcal{R}' = \mathcal{R}$. Then, $\mathcal{G} \subseteq \mathcal{G}'$ and $\mathcal{R} \subseteq \mathcal{R}'$. Therefore, by Corollary 2.19, $\mathcal{I} \rightsquigarrow \mathcal{I}'$. At the same time, since $x \notin \mathcal{R}'$, $\mathcal{G} \parallel \mathcal{R} = \mathcal{G}' \parallel \mathcal{R}'$. Hence, by Definition 2.15 and Definition 2.16, $\mathcal{I} \equiv \mathcal{I}'$. Therefore, by Definition 2.20, $\mathcal{I} \mapsto \mathcal{I}'$. However, $\mathcal{G}' \models_G \mathcal{I}'$, but $\mathcal{G}' \not\models_G \mathcal{I}$, since $\mathcal{G}' \not\subseteq \mathcal{G}$. Therefore, $\mathcal{I} \not\sim \mathcal{I}'$. Thus, by Definition 2.21, $\mathcal{I}$ is not maximal, contradicting our hypothesis.

For the reverse direction, assume interface $\mathcal{I}$ is such that $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$. Let $\mathcal{I}'$ be a completion of $\mathcal{I}$, i.e., $\mathcal{I} \mapsto \mathcal{I}'$. We must show that $\mathcal{I} \sim \mathcal{I}'$, or, by Definition 2.11, that $\mathcal{G} = \mathcal{G}'$ and $\mathcal{R} = \mathcal{R}'$. We proceed as follows. Since $\mathcal{I} \mapsto \mathcal{I}'$, by Definition 2.20, $\mathcal{I} \equiv \mathcal{I}'$ and $\mathcal{I} \rightsquigarrow \mathcal{I}'$. From $\mathcal{I} \rightsquigarrow \mathcal{I}'$ and Corollary 2.19, it follows that $\mathcal{G} \subseteq \mathcal{G}'$ and $\mathcal{R} \subseteq \mathcal{R}'$. Therefore, since $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$, it must also hold that $\mathcal{G}' \cup \mathcal{R}' = \mathcal{B}(\Sigma)$. In addition, from $\mathcal{I} \equiv \mathcal{I}'$, by Definition 2.16 and Definition 2.15, we have $\mathcal{G} \parallel \mathcal{R} = \mathcal{G}' \parallel \mathcal{R}'$. Therefore, by Definition 2.2, $\mathcal{G} \cap \mathcal{R} = \mathcal{G}' \cap \mathcal{R}'$. Assume now that $\mathcal{G}' \not\subseteq \mathcal{G}$. Then, there must be $x \in \mathcal{G}'$ such that $x \notin \mathcal{G}$. Then, since $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$, we must have $x \in \mathcal{R}$. Therefore, since $\mathcal{R} \subseteq \mathcal{R}'$, $x \in \mathcal{R}'$. Consequently, $x \in \mathcal{G}' \cap \mathcal{R}'$. However, since $\mathcal{G} \cap \mathcal{R} = \mathcal{G}' \cap \mathcal{R}'$, it must also be $x \in \mathcal{G} \cap \mathcal{R}$, which contradicts the assumption that $x \notin \mathcal{G}$. Therefore, $\mathcal{G}' \subseteq \mathcal{G}$, and hence, since by the previous argument also $\mathcal{G} \subseteq \mathcal{G}'$, it follows that $\mathcal{G}' = \mathcal{G}$. By similar arguments, we show that $\mathcal{R}' = \mathcal{R}$. Thus, $\mathcal{I} \sim \mathcal{I}'$, and, by Definition 2.21, $\mathcal{I}$ is maximal. □

Corollary 2.23. *An interface $\mathcal{I}$ is maximal if and only if $\mathcal{G} = \mathcal{G} \cup \overline{\mathcal{R}}$.*

Proof. For the forward direction, let $\mathcal{I}$ be maximal. Then, by Theorem 2.22, $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$. Consequently, $\overline{\mathcal{R}} \subseteq \mathcal{G}$ (if $x \notin \mathcal{R}$ then $x \in \mathcal{G}$ for $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$ to hold). Therefore, $\mathcal{G} \cup \overline{\mathcal{R}} = \mathcal{G}$.

For the reverse direction, assume $\mathcal{G} = \mathcal{G} \cup \overline{\mathcal{R}}$ and let $x \in \mathcal{B}(\Sigma)$. There are two cases. If $x \in \mathcal{G}$, then clearly $x \in \mathcal{G} \cup \mathcal{R}$. Assume now $x \notin \mathcal{G}$. Since $\mathcal{G} = \mathcal{G} \cup \overline{\mathcal{R}}$, then also $x \notin \overline{\mathcal{R}}$. Consequently, $x \in \mathcal{R}$. Therefore $x \in \mathcal{G} \cup \mathcal{R}$. Thus, $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$ and, by Theorem 2.22, $\mathcal{I}$ is maximal. □

Theorem 2.24. *Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ be an interface. Then the interfaces $\mathcal{I}^{\mathcal{R}}$ and $\mathcal{I}^{\mathcal{G}}$ defined by*

$$\mathcal{I}^{\mathcal{R}} = (\mathcal{R} \cup \overline{\mathcal{G}}, \mathcal{G}) \quad \text{and} \quad \mathcal{I}^{\mathcal{G}} = (\mathcal{R}, \mathcal{G} \cup \overline{\mathcal{R}})$$

are maximal completions of $\mathcal{I}$.

Proof. Consider $\mathcal{I}^{\mathcal{R}} = (\mathcal{R} \cup \overline{\mathcal{G}}, \mathcal{G})$. Since $\mathcal{G} \cup \mathcal{R} \cup \overline{\mathcal{G}} = \mathcal{B}(\Sigma)$, by Theorem 2.22, $\mathcal{I}^{\mathcal{R}}$ is maximal. In addition, since $\mathcal{G} \cap \overline{\mathcal{G}} = \emptyset$, we have $\mathcal{G} \cap \mathcal{R} = \mathcal{G} \cap (\mathcal{R} \cup \overline{\mathcal{G}})$. Hence, by Definition 2.15 and 2.16, $\mathcal{I} \equiv \mathcal{I}^{\mathcal{R}}$. Considering that $\mathcal{G} \subseteq \mathcal{G}$ and that $\mathcal{R} \subseteq \mathcal{R} \cup \overline{\mathcal{G}}$, by Corollary 2.19 $\mathcal{I} \rightsquigarrow \mathcal{I}^{\mathcal{R}}$. Thus, by Definition 2.20, $\mathcal{I} \mapsto \mathcal{I}^{\mathcal{R}}$. Therefore, $\mathcal{I}^{\mathcal{R}}$ is a maximal completion of $\mathcal{I}$. By similar arguments we show that $\mathcal{I}^{\mathcal{G}}$ is a maximal completion of $\mathcal{I}$. □

Definition 2.25 (Product - Requirement Relaxation). Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be interfaces. Then $\mathcal{I}$ is a product with requirement relaxation of $\mathcal{I}_1$ and $\mathcal{I}_2$, written $\mathcal{I} = \mathcal{I}_1 \parallel_{\mathcal{R}} \mathcal{I}_2$, if and only if

$$\mathcal{G} = \mathcal{G}_1 \cap \mathcal{G}_2 \quad \text{and} \quad \mathcal{R} = (\mathcal{R}_1 \cap \mathcal{R}_2) \cup \overline{(\mathcal{G}_1 \cap \mathcal{G}_2)}.$$

Definition 2.26 (Product - Guarantee Relaxation). Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be interfaces. Then $\mathcal{I}$ is a product with guarantee relaxation of $\mathcal{I}_1$ and $\mathcal{I}_2$, written $\mathcal{I} = \mathcal{I}_1 \parallel_{\mathcal{G}} \mathcal{I}_2$, if and only if

$$\mathcal{G} = (\mathcal{G}_1 \cap \mathcal{G}_2) \cup \overline{(\mathcal{R}_1 \cap \mathcal{R}_2)} \quad \text{and} \quad \mathcal{R} = \mathcal{R}_1 \cap \mathcal{R}_2.$$

Lemma 2.27. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be interfaces. Then $\mathcal{I}_1 \parallel_{\mathcal{R}} \mathcal{I}_2$ and $\mathcal{I}_1 \parallel_{\mathcal{G}} \mathcal{I}_2$ are maximal interfaces.*

Proof. $\mathcal{I}_1 \parallel_{\mathcal{G}} \mathcal{I}_2$ is maximal by Corollary 2.23, since clearly $\mathcal{G} = (\mathcal{G}_1 \cap \mathcal{G}_2) \cup \overline{(\mathcal{R}_1 \cap \mathcal{R}_2)} = \mathcal{G} \cup \overline{\mathcal{R}} = (\mathcal{G}_1 \cap \mathcal{G}_2) \cup \overline{(\mathcal{R}_1 \cap \mathcal{R}_2)} \cup \overline{(\mathcal{R}_1 \cap \mathcal{R}_2)}$.

It is easy to show that Corollary 2.23 holds also if we interchange the roles of $\mathcal{G}$ and $\mathcal{R}$. By similar arguments, then, also $\mathcal{I}_1 \parallel_{\mathcal{R}} \mathcal{I}_2$ is maximal. □

Definition 2.28 (Contract model). The set of maximal interfaces equipped with conformance and product with requirement relaxation forms a *contract model* closed under composition.

3 SYMMETRIES AND VIEWPOINTS

3.1 Weak and strong assumptions

Definition 3.1 (Implication). An *implication* C over alphabet Σ is a pair (A, G), where A (the assumption) and G (the promise) are components over Σ. The implication is equivalent to the component M given by

$$M = A \rightarrow G = G \sqcup \overline{A} = G \cup \overline{A}.$$

3.2 Symmetric interfaces

Definition 3.2 (Dual). Let $C = (A, G)$ be an implication. The dual of C is the implication $\hat{C}$ such that $\hat{C} = (G, A)$.

Definition 3.3 (Symmetric Interface). An interface $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ is symmetric if and only if $\mathcal{R} = \hat{\mathcal{G}}$.

Theorem 3.4. *If $\mathcal{I}$ is a symmetric interface, then $\mathcal{I}$ is also maximal.*

Proof. Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ with $C = (A, G)$. Since $\mathcal{I}$ is symmetric, by Definition 3.3 $\mathcal{R} = \hat{\mathcal{G}}$, and therefore, by Definition 3.2 $\mathcal{R} = (G, A)$. By Definition 3.1, we have $\mathcal{G} = G \cup \overline{A}$ and $\mathcal{R} = A \cup \overline{G}$. Let us now compute $\mathcal{G} \cup \overline{\mathcal{R}}$. By De Morgan's Law, $\overline{\mathcal{R}} = \overline{(A \cup \overline{G})} = (\overline{A} \cap G)$. Thus, $\overline{\mathcal{R}} \subseteq G$. Therefore, $\mathcal{G} \cup \overline{\mathcal{R}} = G \cup \overline{A} \cup (\overline{A} \cap G) = G \cup \overline{A} = \mathcal{G}$. Hence, by Corollary 2.23, $\mathcal{I}$ is maximal. □

Theorem 3.5. *Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ be a maximal interface. Then there exists a symmetric interface $\mathcal{I}' = ((G, A), (A, G))$ such that $\mathcal{I} \sim \mathcal{I}'$.*

Proof. It is sufficient to assign $A = \mathcal{R}$ and $G = \mathcal{G}$. In this case, by Definition 3.1, $\mathcal{I}'$ is the same as $\mathcal{I}' = (\mathcal{R} \cup \overline{\mathcal{G}}, \mathcal{G} \cup \overline{\mathcal{R}})$. Because $\mathcal{I}$ is maximal, by Theorem 2.22 $\mathcal{G} \cup \mathcal{R} = \mathcal{B}(\Sigma)$. Hence, $\overline{\mathcal{G}} \subseteq \mathcal{R}$, which implies that $\mathcal{R} \cup \overline{\mathcal{G}} = \mathcal{R}$. Similarly, $\overline{\mathcal{R}} \subseteq \mathcal{G}$, which implies that $\mathcal{G} \cup \overline{\mathcal{R}} = \mathcal{G}$. Therefore, $\mathcal{I}' = (\mathcal{R}, \mathcal{G})$.

Several other assignments can be used to obtain a symmetric interfaces, such as the following:

$$A = \overline{\mathcal{G}}, \quad G = \overline{\mathcal{R}}. \qquad A = \overline{\mathcal{G}} \cup \mathcal{R}, \quad G = \overline{\mathcal{R}}. \qquad A = \overline{\mathcal{G}}, \quad G = \overline{\mathcal{R}} \cup \mathcal{G}.$$

or other variations in between. □

Definition 3.6 (Symmetric canonical form). A symmetric interface $\mathcal{I} = (\hat{C}, C)$, with $C = (A, G)$ is in *canonical form* whenever $A = A \cup \overline{G}$ and $G = G \cup \overline{A}$.

Theorem 3.7 (Conformance of symmetric interfaces in canonical form). *Let $\mathcal{I} = (\hat{C}, C)$, and $\mathcal{I}' = (\hat{C}', C')$ be symmetric interfaces in canonical form, with $C = (A, G)$ and $C' = (A', G')$. Then, $\mathcal{I} \preceq \mathcal{I}'$ if and only if*

$$G \subseteq G' \quad \textit{and} \quad A' \subseteq A.$$

Proof. Consider first the guarantees. Since $\mathcal{I}$ and $\mathcal{I}'$ are in canonical form, by Definition 3.6, $G \subseteq G'$ is the same as $G \cup \overline{A} \subseteq \mathcal{G}' \cup \overline{A'}$. Thus, by Definition 3.1, this is the same as $C \subseteq C'$. Therefore, by Definition 2.5, that condition is equivalent to $C \preceq C'$. The same reasoning applied to the assumptions shows that $A' \subseteq A$ is the same as $\hat{C}' \preceq \hat{C}$. The result then follows from Definition 2.11. □

3.3 A revised notion of contract merging

Definition 3.8 (Merging). Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be contracts. Then $\mathcal{I}$ is the resulting of merging $\mathcal{I}_1$ and $\mathcal{I}_2$, written $\mathcal{I} = \mathcal{I}_1 \cdot \mathcal{I}_2$, if and only if

$$\mathcal{G} = (\mathcal{G}_1 \cap \mathcal{G}_2) \cup \overline{(\mathcal{R}_1 \cap \mathcal{R}_2)} \quad \text{and} \quad \mathcal{R} = \mathcal{R}_1 \cap \mathcal{R}_2.$$

3.4 Composition, merging, and the contract lattice

Theorem 3.9. *Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ and $\mathcal{I}' = (\mathcal{R}', \mathcal{G}')$ be contracts. Then the contract $\mathcal{I} \sqcap \mathcal{I}'$ is equal to the conjunction of the following three contracts:*

(1) $(\mathcal{R} - \mathcal{R}', \mathcal{G} \cup \overline{\mathcal{R} - \mathcal{R}'})$
(2) $(\mathcal{R}' - \mathcal{R}, \mathcal{G}' \cup \overline{\mathcal{R}' - \mathcal{R}})$
(3) $(\mathcal{R} \cap \mathcal{R}', \mathcal{G} \cap \mathcal{G}' \cup \overline{\mathcal{R} \cap \mathcal{R}'})$

Proof. The union of the requirements of the three contracts clearly yields $\mathcal{R} \cup \mathcal{R}'$. We compute the intersection of the guarantees:

$$\begin{aligned}
&(\mathcal{G} \cup \mathcal{R}') \cap (\mathcal{G}' \cup \mathcal{R}) \cap (\mathcal{G} \cap \mathcal{G}' \cup \overline{\mathcal{R} \cap \mathcal{R}'}) \\
&\quad = (\mathcal{G} \cup \mathcal{R}') \cap \Big((\mathcal{G} \cap \mathcal{G}') \cup (\mathcal{G}' \cap \overline{\mathcal{R}}) \cup (\mathcal{G}' \cap \overline{\mathcal{R}'}) \cup (\mathcal{R} \cap \overline{\mathcal{R}'}) \Big) \\
&\quad = (\mathcal{G} \cup \mathcal{R}') \cap \Big((\mathcal{G} \cap \mathcal{G}') \cup \overline{\mathcal{R}'} \Big) \quad (\text{since } \mathcal{G} \cap \overline{\mathcal{R}'} \preceq \mathcal{G} \cap \mathcal{G}') \\
&\quad = \Big((\mathcal{G} \cap \mathcal{G}') \cup (\mathcal{G} \cap \overline{\mathcal{R}'}) \Big) = \mathcal{G} \cap \mathcal{G}',
\end{aligned}$$

which are the guarantees of the conjunction. □

Theorem 3.10. *Let $\mathcal{I} = (\mathcal{R}, \mathcal{G})$ and $\mathcal{I}' = (\mathcal{R}', \mathcal{G}')$ be contracts. Then the contract $\mathcal{I} \sqcup \mathcal{I}'$ is equal to the disjunction of the following three contracts:*

(1) $(\mathcal{R} \cup \overline{\mathcal{G} - \mathcal{G}'}, \mathcal{G} - \mathcal{G}')$
(2) $(\mathcal{R}' \cup \overline{\mathcal{G}' - \mathcal{G}}, \mathcal{G}' - \mathcal{G})$
(3) $(\mathcal{R} \cap \mathcal{R}' \cup \overline{\mathcal{G} \cap \mathcal{G}'}, \mathcal{G} \cap \mathcal{G}')$

Proof. Obtained by dualizing the proof of Theorem 3.9. □

3.5 Decomposition of contracts and separation of viewpoints

THEOREM 3.11 (CONTRACT QUOTIENT). *Let $\mathcal{I}$ and $\mathcal{I}_1$ be contracts. Then $\mathcal{I}_q$ is the quotient between contracts $\mathcal{I}$ and $\mathcal{I}_1$, written $\mathcal{I}_q = \mathcal{I} \,/\, \mathcal{I}_1$, if and only if*

$$\mathcal{G}_q = \mathcal{G} \cap \mathcal{R}_1 \cup \overline{(\mathcal{R} \cap \mathcal{G}_1)} \quad \textit{and} \quad \mathcal{R}_q = \mathcal{R} \cap \mathcal{G}_1.$$

PROOF. We wish to show that $\mathcal{I}_q$ satisfies

$$\forall \mathcal{I}'. \quad \mathcal{I}' \parallel \mathcal{I}_1 \preceq \mathcal{I} \iff \mathcal{I}' \preceq \mathcal{I}_q. \tag{1}$$

Suppose $\mathcal{I}'$ is a contract that conforms to $\mathcal{I}_q$. Let $\mathcal{I}_2 = \mathcal{I}_1 \parallel \mathcal{I}_q$. Expanding, we have

$$\mathcal{G}_2 = \mathcal{G}_1 \cap (\mathcal{R}_1 \cap \mathcal{G} \cup \overline{\mathcal{R}}) \quad \text{and}$$
$$\mathcal{R}_2 = \mathcal{R} \cup \overline{\mathcal{G}_1}.$$

Clearly, $\mathcal{G}_2 \preceq \mathcal{G}$ and $\mathcal{R} \preceq \mathcal{R}_2$, so $\mathcal{I}_2 \preceq \mathcal{I}$. Since composition is monotonic with respect to conformance, $\mathcal{I}' \parallel \mathcal{I}_1 \preceq \mathcal{I}_2 \preceq \mathcal{I}$.

Conversely, suppose $\mathcal{I}'' \preceq \mathcal{I}$ for $\mathcal{I}'' = \mathcal{I}' \parallel \mathcal{I}_1$. We wish to show that $\mathcal{I}' \preceq \mathcal{I}_q$. From the assumption $\mathcal{I}'' \preceq \mathcal{I}$, we observe that

$$\mathcal{R} \preceq \mathcal{R}'' = \mathcal{R}' \cap \mathcal{R}_1 \cup \overline{(\mathcal{G}' \cap \mathcal{G}_1)} \preceq \mathcal{R}' \cup \overline{\mathcal{G}_1}$$
$$\therefore \quad \mathcal{R}_q = \mathcal{R} \cap \mathcal{G}_1 \preceq \mathcal{R}'. \tag{2}$$

Rewriting the left-hand side of (2), we have

$$\overline{(\mathcal{R}' \cap \mathcal{R}_1)} \cap (\mathcal{G}' \cap \mathcal{G}_1) \preceq \overline{\mathcal{R}}$$
$$\therefore \quad \mathcal{G}' \preceq \overline{\mathcal{R}} \cup (\mathcal{R}' \cap \mathcal{R}_1) \cup \overline{\mathcal{G}_1} \preceq \overline{\mathcal{R}} \cup \mathcal{R}_1. \tag{3}$$

From the hypothesis $\mathcal{I}'' \preceq \mathcal{I}$, we have $\mathcal{G}' \cap \mathcal{G}_1 \preceq \mathcal{G}$. This constraint together with (3) gives us

$$\mathcal{G}' \preceq (\overline{\mathcal{R}} \cup \mathcal{R}_1) \cap (\mathcal{G} \cup \overline{\mathcal{G}_1}) = \mathcal{G} \cap \mathcal{R}_1 \cup \overline{(\mathcal{R} \cap \mathcal{G}_1)} = \mathcal{G}_q.$$

This result and (2) imply that $\mathcal{I}' \preceq \mathcal{I}_q$. □

THEOREM 3.12 (SEPARATION). *Let $\mathcal{I}$ and $\mathcal{I}_1$ be contracts. Then $\mathcal{I}_s$ is the separation of contracts $\mathcal{I}$ and $\mathcal{I}_1$, written $\mathcal{I}_s = \mathcal{I} \div \mathcal{I}_1$, if and only if*

$$\mathcal{G}_s = \mathcal{G} \cap \mathcal{R}_1 \quad \textit{and} \quad \mathcal{R}_s = \mathcal{R} \cap \mathcal{G}_1 \cup \overline{(\mathcal{G} \cap \mathcal{R}_1)}.$$

PROOF. The proof is obtained by dualizing the proof of Theorem 3.11. We give the full details.

We wish to show that $\mathcal{I}_s$ satisfies

$$\forall \mathcal{I}'. \quad \mathcal{I} \preceq \mathcal{I}' \cdot \mathcal{I}_1 \iff \mathcal{I}_s \preceq \mathcal{I}'.$$

Suppose $\mathcal{I}_s \preceq \mathcal{I}'$ for a contract $\mathcal{I}'$. Let $\mathcal{I}_2 = \mathcal{I}_1 \cdot \mathcal{I}_s$. Expanding, we have

$$\mathcal{R}_2 = \mathcal{R}_1 \cap (\mathcal{G}_1 \cap \mathcal{R} \cup \overline{\mathcal{G}}) \quad \text{and}$$
$$\mathcal{G}_2 = \mathcal{G} \cup \overline{\mathcal{R}_1}.$$

Clearly, $\mathcal{R}_2 \preceq \mathcal{R}$ and $\mathcal{G} \preceq \mathcal{G}_2$, so $\mathcal{I} \preceq \mathcal{I}_2$. Since merging is monotonic with respect to conformance, $\mathcal{I} \preceq \mathcal{I}_2 \preceq \mathcal{I}' \parallel \mathcal{I}_1$.

Conversely, suppose $\mathcal{I} \preceq \mathcal{I}''$ for $\mathcal{I}'' = \mathcal{I}' \cdot \mathcal{I}_1$. We wish to show that $\mathcal{I}_s \preceq \mathcal{I}'$. From the assumption $\mathcal{I} \preceq \mathcal{I}''$, we observe that

$$\mathcal{G} \preceq \mathcal{G}'' = \mathcal{G}' \cap \mathcal{G}_1 \cup \overline{(\mathcal{R}' \cap \mathcal{R}_1)} \preceq \mathcal{G}' \cup \overline{\mathcal{R}_1}$$
$$\therefore \quad \mathcal{G}_s = \mathcal{G} \cap \mathcal{R}_1 \preceq \mathcal{G}'. \tag{4}$$

Rewriting the left-hand side of (4), we have

$$\overline{(\mathcal{G}' \cap \mathcal{G}_1)} \cap (\mathcal{R}' \cap \mathcal{R}_1) \preceq \overline{\mathcal{G}}$$
$$\therefore \quad \mathcal{R}' \preceq \overline{\mathcal{G}} \cup (\mathcal{G}' \cap \mathcal{G}_1) \cup \overline{\mathcal{R}_1} \preceq \overline{\mathcal{G}} \cup \mathcal{G}_1. \tag{5}$$

From the hypothesis $\mathcal{I} \preceq \mathcal{I}''$, we have $\mathcal{R}' \cap \mathcal{R}_1 \preceq \mathcal{R}$. This constraint together with (5) gives us

$$\mathcal{R}' \preceq (\overline{\mathcal{G}} \cup \mathcal{G}_1) \cap (\mathcal{R} \cup \overline{\mathcal{R}_1}) = \mathcal{R} \cap \mathcal{G}_1 \cup \overline{(\mathcal{G} \cap \mathcal{R}_1)} = \mathcal{R}_s.$$

This result and (4) imply that $\mathcal{I}_s \preceq \mathcal{I}'$. □

Definition 3.13 (Composition and merging identity). A composition identity, denoted $\mathbb{1}_c$, is a contract such that $\mathcal{I} \parallel \mathbb{1}_c = \mathbb{1}_c \parallel \mathcal{I} = \mathcal{I}$. Likewise, a merging identity, denoted $\mathbb{1}_m$, satisfies $\mathcal{I} \cdot \mathbb{1}_m = \mathbb{1}_m \cdot \mathcal{I} = \mathcal{I}$.

Lemma 3.14. *Let Σ be the union of all alphabets over which components are defined. The contract identities just introduced and the contracts $\bot$ and $\top$ have the following explicit forms: $\top = (\emptyset, \mathcal{B}(\Sigma))$, $\bot = (\mathcal{B}(\Sigma), \emptyset)$, and $\mathbb{1}_m = \mathbb{1}_c = (\mathcal{B}(\Sigma), \mathcal{B}(\Sigma))$. Since both identities are equal, we call $\mathbb{1} = \mathbb{1}_c = \mathbb{1}_m$ the identity.*

Proof. Obvious. □

Definition 3.15 (Reciprocal). Let $I = (\mathcal{R}, \mathcal{G})$. Its reciprocal, denoted I^{-1}, is given by $I^{-1} = \mathbb{1} \,/\, \mathcal{I} = \mathbb{1} \div \mathcal{I} = (\mathcal{G}, \mathcal{R})$.

3.6 Multiviewpoint design

4 RELATED WORK

In this report we discuss the formal underpinnings for abstraction and reuse in the design of embedded, hybrid, heterogeneous and cyber-physical systems for early system and software design and verification [20, 72, 73]. In particular, determining the correctness of designs made of library parts that are developed independently implies the development of formal methods that can guarantee compositionality. In our previous work, we have addressed several of these issues [6, 17, 30, 66, 69, 71, 80, 81]. This must be ensured in the presence of heterogeneous models and multiple viewpoint design, which we discuss in particular in our previous work related to heterogeneous design methods [12, 15, 16, 50, 52, 54, 66]. While this paper deals primarily with theoretical results with regard to contract model operators, we have applied several of the ideas in methodologies and tools [11, 18, 25, 38, 39, 44, 45, 61, 62, 67], including both simulation methods, and design space exploration [3–5, 28, 36, 60], applied to a number of case studies [7, 14, 19, 31, 51, 68, 77].

Here, we focus in particular on contract models. In previous work, we have addressed some of the concerns discussed here using modalities and special operators [9, 10, 35, 74–76, 78] and we have analyzed the problem of correct decomposition [49, 53]. The results in this report extend this work, providing new and coherent operators for both hierarchical and viewpoint decomposition.

The notion of contract as used in our framework derives from the theory of abstract data types and was first suggested by Meyer in the context of the programming language Eiffel [59], following the original ideas introduced by Floyd and Hoare [37, 41] to assign logical meaning to sequential imperative programs in the form of triples of assertions. In his work, Meyer introduces *preconditions* and *postconditions* as assertions or specifications for the methods of a class, and *invariants* for the class itself. Preconditions correspond to the assumptions under which the method operates, while postconditions express the promises at method termination, provided that

the assumptions are satisfied. Promises must be guaranteed only if the assumptions are satisfied. Invariants, on the other hands, are conditions that must be true of the state of the class regardless of any assumption. The notion of class inheritance, in this case, is used as a refinement, or sub-typing, relation. To guarantee safe substitutability, a subclass is only allowed to weaken assumptions and to strengthen promises and invariants. Similar ideas were already present in seminal work by Dijkstra [32] and Lamport [46] on *weakest preconditions* and *predicate transformers* for sequential and concurrent programs, and in more recent work by Back and von Wright, who introduce contracts in the *refinement calculus* [2]. In this formalism, processes are described with guarded commands operating on shared variables. Contracts are composed of *assertions* (higher-order state predicates) and *state transformers*. These contracts are of a very different nature, since there is no clear indication of the role (assumption or promise) a state predicate or a state transformer may play. This formalism is best suited to reason about discrete, un-timed process behavior.

Most interface and contract models fit within our theory, and their properties can be well explained by our results. Besides the ones that were already mentioned, three in particular form the basis of the models that were subsequently developed in the literature. Our work is based, in particular, on three models that were subsequently developed in the literature. The work of Dill on asynchronous trace structures was the first to differentiate between acceptable and non-acceptable uses of a component [33]. Behaviors, or traces, can be either accepted as *successes*, or rejected as *failures*. The failures, which are still possible behaviors of the system, correspond to unacceptable inputs from the environment, and are therefore the complement of the assumptions. Safe substitutability is expressed as trace containment between the successes and failures of the specification and the implementation. The conditions obtained by Dill are equivalent to requiring that the implementation weaken the assumptions of the specification while strengthening the promises. If we denote the set of successful traces by S, and by F the set of failure traces, one obtains a maximal interface $\mathcal{I}$ by setting

$$\mathcal{G} = S \cup F, \quad \mathcal{R} = \overline{F}.$$

The notion of refinement and composition are the same as for maximal interfaces, where composition is taken as the product with requirement relaxation. We have shown in this paper how this is only one possible choice for these operators, by working in a more general settings. The relaxation mechanism is obtained through a process called *autofailure manifestation* and *failure exclusion*, which yield a maximal interface. Wolf later extended the same technique to a discrete synchronous model [85]. The trace structures developed by Dill and Wolf are particularly interesting because they address the problem of receptiveness, or input completeness, by proving closure properties and by giving decision procedures. Finally, Process Spaces [63] is a more general model proposed by Negulescu following the work of Dill and Wolf, and is based on processes equivalent to our maximal interfaces with two sets: X is the set of possible behaviors, and Y is the set of acceptable behaviors. A process has the requirement that $X \cup Y = \mathcal{B}$, making it equivalent to a maximal interface. A process is the same as an interface $\mathcal{I}$ that has:

$$\begin{aligned} M_{\mathcal{I}} &= X, \\ E^{\mathcal{I}} &= Y. \end{aligned}$$

So, processes correspond to maximal interfaces. The notion of conformance corresponds to the one proposed here. While several operators are introduced, Process Spaces ignore the issues with inverse projection in conjunction, and weak and strong assumptions. Process Spaces define the operations of product and exclusive sum, which are syntactically the same as our operations of parallel composition and merging for contracts, respectively. Product is used to compose design elements, but no insight is given into the use of exclusive sum. In particular, the paper does not address multiple viewpoints for the same design element. Also, this work does not show how these operations relate to the lattice operations generated by the conformance order, as we do in Section 3.4, and no distinction is made between inputs and outputs. In process spaces, an operation called reflection is defined.

This operation has the same closed form as the operation of reciprocal that we introduced. Nonetheless, while Negulescu *defines* the closed-form expression for reflection, in our development the closed-form of the reciprocal is derived from an expression involving the identity and the quotient. The classic Interface Automata [29] and HRC [12, 27] models are similar to synchronous trace structures, where failures are implicitly all the traces that are not accepted. Thus, the interface is maximal. Composition is defined on automata, rather than on traces, and requires a procedure similar to requirement relaxation (and therefore to autofailure manifestation) in order to maintain maximality. The authors have also extended the approach to several other kinds of behaviors, including resources and asynchronous behaviors [22, 40]. We take these concepts a lot further in this paper to include several other operators.

These concepts were subsequently developed to maturity by giving rise to MDE (*Model Driven Engineering*) [43, 55, 79]. In this context, interfaces are described as part of the system architecture and comprise typed ports, parameters and attributes. Contracts on interfaces are typically formulated in terms of constraints on the entities of components, using the Object Constraint Language (OCL) [65, 84]. Roughly speaking, an OCL statement refers to a context for the considered statement, and expresses properties to be satisfied by this context (e.g., if the context is a class, a property might be an attribute). Arithmetic or set-theoretic operations can be used in expressing these properties. To account for behavior and performance, the classical approach in MDE consists in enriching components with *methods* that can be invoked from outside, and/or *state machines*. Attributes on port methods have been used to represent non-functional requirements or provisions of a component [18]. The effect of a method is made precise by the actual code that is executed when calling this method. The state machine description and the methods together provide directly an implementation for the component — actually, several MDE related tools, such as GME and Rational Rose, automatically generate executable code from this specification. The notion of refinement is replaced by the concept of class inheritance. Inheritance, however, is unable to cover aspects related to behavior refinement, since it is limited to constraining the signature of the method, rather than their behavior. Nor is it made precise what it means to take the conjunction of interfaces, only approximated by multiple inheritance, or to compose them. Liskov and Wing [57] address some of these shortcomings by strengthening Meyer's contract model to include the specification of extra methods in the subtype, which may change the object state, and to account for the preservation of history properties.

Others in the literature have approached the problem of studying the properties of contract models from an abstract point of view. One of the early foundational work that concerns interface and contract specification is due to Abadi and Lamport, who were first to thoroughly discuss and differentiate between the component guarantees and the environment assumptions [1]. In this work, the authors focus on the formulation of the specification as an *implication*, in the sense described in Section 3.1. In particular, while a specification is allowed to make assumptions, it is *not* interpreted as constraining the environment, or else the specification is considered unrealizable. In this paper we formalize the difference between these two interpretations as weak and strong assumptions. At the same time, however, and following the work of Dill [33], realizability is defined as a game, a technique that would later be employed in other popular interface models, such as interface automata [29]. Of greater significance, when defining the refinement relation, the authors insist that assumptions be weakened, which is unnecessary for implications, thus effectively obtaining the equivalent of an interface model. The main result of Abadi and Lamport is a full set of proof rules that show when the parallel composition of components satisfies a given property, under a set of assumptions. These proof rules have later been reformulated in similar ways in several other contract models and tools [8, 24, 34, 38]. Composition, expressed as intersection of behaviors, takes primarily the view of the component. Our work builds on these concepts, and evolves in complementary directions. First, we emphasize an approach to composition that is balanced between component and environment, leading to the notions of maximal and symmetric interfaces. More importantly, while Abadi and Lamport touch upon the need for inverse projection during composition, they do not discuss the operation of conjunction and all that it entails. They also discusses the validity of the circular reasoning principle when liveness properties are

considered. Our work is largely orthogonal to these aspects, although intersection of behaviors could clearly lead to no or multiple solutions. Dill uses failures in infinite traces to express causality and liveness [33], a method that could be reflected in contracts. Also, proof rules similar to those proposed by Graf et al. [38] in HRC could be employed to determine how and when circular reasoning allows a component and its environment to be refined concurrently, each relying on the abstract description of its context, and therefore prove conformance.

Bauer et al. [8] present a meta-theory similar in spirit to our work. The objective is to provide a method with which to construct a *contract* framework given a *specification* (or component, with our terminology) framework with sufficient reasonable properties. The idea is to devise a set of axiomatic definitions for the contract operators and relations, with provable compositional properties, which can be instantiated given a specification theory. The work focuses on the relation of refinement and defines operators for composition, conjunction and quotient. In particular, they show how to constructively define the composition operator. Their method is based on the use of canonical forms, and treats environments and implementations asymmetrically. The modal specification model introduced by Raclet et al. [75] is used as a case study. We follow a similar approach, and start from a generic component model to build interface and contract models with increasing levels of structure. Our objectives, however, are complementary. We focus in particular on dissecting the interface models to thoroughly understand the role of each of its parts, and determine their expressiveness by considering different notions of refinement and equivalence. Unlike the cited work, we treat environments and implementations on an equal ground, to include methodologies that favor both component optimization and component reuse. We show the pitfalls associated with applying inverse projection and union and propose an effective solution, using merging (see Section 3.3). In particular, we do so from a purely semantic point of view, instead of employing the traditional transition systems.

Chilton et al. [23] develop an algebraic theory of interface automata based on traces which is also useful to shed light on the properties of the operators and relations. The formalism is reminiscent of Dill's trace structures, and extends that work with additional operators, such as quotient. The authors also address issues of progress in the context of finite traces, unlike trace structures which use infinite traces. Of particular interest is the definition of refinement, which allows the refining component to have signatures with different sets of inputs and outputs. Consequently, conjunction can also be defined on components with different signatures. This facilitates a multiple viewpoint approach. However, the issue of inverse projection is not resolved, and the conjunction "yields the coarsest component that will work in any environment safe for at least one of its operands" [23]. Hence, a viewpoint that makes no assumption will necessarily wipe out the assumptions of all other viewpoints, as discussed in Section 3.3. We believe the merging operator that we introduce could be applied to their model to properly handle these cases.

Most recently, Benveniste et al. have introduced a meta-theory of contracts to frame several models in the same formalism [10], and discuss their operators. We also reason about the contract operators in a general settings, and cover aspects, such as completion and merging, which were not analyzed previously. Our aim, however, is not to construct another or better meta-theory. Instead, the formalism that we use is largely derived from previous models (such as the mentioned HRC) and has been adapted to simplify the algebraic expression of our main results. The quotient operation we discussed is the same as that defined in terms of its universal property (1) in Benveniste et al. [10]. The closed-form expression for the quotient operation of contracts was introduced in [42]. As far as we know, the separation operation we discussed has no precedent in the literature.

Tripakis et al [83] also study the connection between different kinds of interface specification. In particular, they show how to transform Relational Interfaces [82], which are not input complete (or receptive), into an equivalent set of input complete specifications, in order to avoid game-theoretic methods and have a more efficient analysis. We believe this procedure is akin to going from Interface Automata to Trace Structures. Similarly, Carmona and Kleijn [21] explore the issue of compatibility in a general multi-component settings. This work deals primarily with questions of receptiveness, progress and deadlock freedom. However, the authors do not develop a full interface or contract model, but express assumptions implicitly in terms of the actions which are enabled at each

state of the components. An analysis of these aspects, which are orthogonal to the work presented in this paper, and their application to our context are part of our future work.

Damm et al. [26] introduce the distinction between weak and strong assumptions, which we extend using the concept of implication. Mangeruca et al. [58] use a similar notion, called *precondition*, to define the conditions under which the promises must hold, in a form similar to implications. The authors use this concept to define the *completeness* of a contract relative to the requirements, and avoid implementations that vacuously satisfy their contract. The formalism is also used to define extensions of the contract, by properly combining the promises and their preconditions. This differs from our notion of completion, and is used to help designers cope with evolving specifications. The authors also provide an operator to override a promise by another promise. These extensions naturally fit in our formalism when promises are defined as implications. Using our theory, the same operators can similarly be extended to cover also the environment requirements.

5 CONCLUSIONS

This report extends with proofs the theory presented at EMSOFT 2019 [70].

ACKNOWLEDGMENTS

This work was supported in part by NSF Contract CPS Medium 1739816.

REFERENCES

[1] Martín Abadi and Leslie Lamport. 1993. Composing Specifications. *ACM Transactions on Programming Languages and Systems* 15, 1 (January 1993), 73–132.

[2] Ralph-Johan Back and Joakim von Wright. 2000. Contracts, Games, and Refinement. *Information and communication* 156 (2000), 25–45.

[3] Felice Balarin, Abhijit Davare, Massimiliano D'Angelo, Douglas Densmore, Trevor Meyerowitz, Roberto Passerone, Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Alena Simalatsar, Yoshinori Watanabe, Guang Yang, and Qi Zhu. 2009. Platform-Based Design and Frameworks: Metropolis and Metro II. In *Model-Based Design for Embedded Systems*, Gabriela Nicolescu and Pieter J. Mosterman (Eds.). CRC Press, Taylor and Francis Group, Boca Raton, London, New York, Chapter 10, 259.

[4] Felice Balarin and Roberto Passerone. 2006. Functional Verification Methodology Based on Formal Interface Specification and Transactor Generation. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE06)*. European Design and Automation Association, 3001 Leuven, Belgium, Munich, Germany, 1013–1018.

[5] Felice Balarin and Roberto Passerone. 2007. Specification, Synthesis and Simulation of Transactor Processes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 10 (October 2007), 1749–1762.

[6] Felice Balarin, Roberto Passerone, Alessandro Pinto, and Alberto L. Sangiovanni-Vincentelli. 2005. A Formal Approach to System Level Design: Metamodels and Unified Design Environments. In *Proceedings of the Third ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE05)*. IEEE Computer Society, Los Alamitos, CA, USA, Verona, Italy, 155–163.

[7] Alexander Baranov, Denis Spirjakin, Saba Akbari, Andrey Somov, and Roberto Passerone. 2016. POCO: 'Perpetual' Operation of CO Sensor Node with Hybrid Power Supply. *Sensors & Actuators A: Physical* 238 (2016), 112–121.

[8] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. 2012. Moving from specifications to contracts in component-based design. In *Proceedings of the* 15^{th} *International Conference on Fundamental Approaches to Software Engineering (FASE'12)*. Springer-Verlag, Tallinn, Estonia, 43–58.

[9] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. 2008. Multiple Viewpoint Contract-Based Specification and Design. In *Formal Methods for Components and Objects,* 6^{th} *International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007, Revised Papers*, Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever (Eds.). Lecture Notes in Computer Science, Vol. 5382. Springer Verlag, Berlin Heidelberg, 200–225.

[10] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. 2018. *Contracts for System Design*. Foundations and Trends in Electronic Design Automation, Vol. 12. now publishers. 124–400 pages.

[11] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. 2009. Multi-Viewpoint State Machines for Rich Component Models. In *Model-Based Design for Embedded Systems*, Gabriela Nicolescu and Pieter J. Mosterman (Eds.). CRC Press, Taylor and Francis Group, Boca Raton, London, New York, Chapter 15, 487.

[12] Luca Benvenuti, Alberto Ferrari, Leonardo Mangeruca, Emanuele Mazzi, Roberto Passerone, and Christos Sofronis. 2008. A Contract-Based Formalism for the Specification of Heterogeneous Systems. In *Proceedings of the Forum on Specification, Verification and Design*

Languages (FDL08). Stuttgart, Germany, 142–147.

[13] Luca Benvenuti, Alberto Ferrari, Emanuele Mazzi, and Alberto L. Sangiovanni Vincentelli. 2008. Contract-Based Design for Computation and Verification of a Closed-Loop Hybrid System. In *Hybrid Systems: Computation and Control*, Magnus Egerstedt and Bud Mishra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 58–71.

[14] Davide Brunelli, Ivan Minakov, Roberto Passerone, and Maurizio Rossi. 2014. POVOMON: an Ad-hoc Wireless Sensor Network for Indoor Environmental Monitoring. In *Proceedings of the 2014 IEEE Workshop on Environmental, Energy and Structural Monitoring Systems (EESMS14)*. IEEE, Naples, Italy, 1–6.

[15] Jerry R. Burch, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2001. Overcoming Heterophobia: Modeling Concurrency in Heterogeneous Systems. In *Proceedings of the* 2^{nd} *International Conference on Application of Concurrency to System Design (ACSD01)*. IEEE Computer Society, Los Alamitos, CA, USA, Newcastle upon Tyne, UK, 13–32.

[16] Jerry R. Burch, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2001. Using Multiple Levels of Abstractions in Embedded Software Design. In *First International Workshop on Embedded Software, EMSOFT01 (Lecture Notes in Computer Science)*, Thomas A. Henzinger and Christoph M. Kirsch (Eds.), Vol. 2211. Springer, Tahoe City, CA, USA, 324–343.

[17] Jerry R. Burch, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2002. Modeling Techniques in Design-by-Refinement Methodologies. In *Proceedings of the Forum on Specification and Design Languages, invited paper (FDL02)*. Marseille, France.

[18] Daniela Cancila, Roberto Passerone, Tullio Vardanega, and Marco Panunzio. 2010. Toward Correctness in the Specification and Handling of Non-Functional Attributes of High-Integrity Real-Time Embedded Systems. *IEEE Transactions on Industrial Informatics* 6, 2 (May 2010), 181–194.

[19] Daniela Cancila, Elie Soubiran, and Roberto Passerone. 2014. Feasibility Study in the Use of Contract-Based Approaches to Deal with Safety-Related Properties in CPS. *Ada User Journal* 35, 4 (December 2014), 272–277.

[20] Luca P. Carloni, Roberto Passerone, Alessandro Pinto, and Alberto L. Sangiovanni-Vincentelli. 2006. *Languages and Tools for Hybrid Systems Design*. Foundations and Trends in Electronic Design Automation, Vol. 1. now publishers. 1–193 pages.

[21] Josep Carmona and Jetty Kleijn. 2013. Compatibility in a multi-component environment. *Theoretical Computer Science* 484 (May 2013), 1–15.

[22] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Marielle Stoelinga. 2003. Resource Interfaces. In *Proceedings of the Third Annual Conference on Embedded Software (EMSOFT03) (Lecture Notes in Computer Science)*, Vol. 2855. Springer, 117–133.

[23] Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. 2014. An algebraic theory of interface automata. *Theoretical Computer Science* 549 (September 2014), 146–174.

[24] Alessandro Cimatti and Stefano Tonetta. 2015. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming* 97, Part 3 (2015), 333–348.

[25] Loris Dal Lago, Orlando Ferrante, Roberto Passerone, and Alberto Ferrari. 2018. Dependability Assessment of SOA-based CPS with Contracts and Model-Based Fault Injection. *IEEE Transactions on Industrial Informatics* 14, 1 (January 2018), 360–369.

[26] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. 2011. Using contract-based component specifications for virtual integration testing and architecture design. In *Design, Automation Test in Europe Conference Exhibition (DATE11)*. Grenoble, France, 1–6.

[27] W. Damm, A. Votintseva, A. Metzner, B. Josko, P. Peikenkamp, and E. Bode. 2005. Boosting re-use of embedded automotive applications through rich components. In *Foundations of Interface Technologies (FIT'05)*.

[28] Abhijit Davare, Douglas Densmore, Liangpeng Guo, Roberto Passerone, Alberto L. Sangiovanni-Vincentelli, Alena Simalatsar, and Qi Zhu. 2013. METROII: A Design Environment for Cyber-Physical Systems. *ACM Transactions on Embedded Computing Systems* 12, 1s (March 2013), 49:1–49:31.

[29] Luca de Alfaro and Thomas A. Henzinger. 2001. Interface Automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*. ACM Press, 109–120.

[30] Douglas Densmore, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2006. A Platform-Based Taxonomy for ESL Design. *IEEE Design and Test of Computers* 23, 5 (May 2006), 359–374.

[31] Douglas Densmore, Alena Simalatsar, Abhijit Davare, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. 2009. UMTS MPSoC Design Evaluation Using a System Level Design Framework. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE09)*. Nice, France, 478–483.

[32] Edsger W. Dijkstra. 1975. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM* 18, 8 (August 1975), 453–457.

[33] David L. Dill. 1989. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT Press.

[34] Iulia Dragomir, Iulian Ober, and Christian Percebois. 2015. Contract-based modeling and verification of timed safety requirements within SysML. *Software & Systems Modeling* (2015), 1–38.

[35] Orlando Ferrante, Roberto Passerone, Alberto Ferrari, Leonardo Mangeruca, and Christos Sofronis. 2014. BCL: a compositional contract language for embedded systems. In *Proceedings of the* 19^{th} *IEEE International Conference on Emerging Technologies and Factory Automation*

(ETFA14). Barcelona, Spain, 1–6.

[36] Orlando Ferrante, Roberto Passerone, Alberto Ferrari, Leonardo Mangeruca, Christos Sofronis, and Massimiliano D'Angelo. 2014. Monitor-Based Run-Time Contract Verification of Distributed Systems. In *Proceedings of the* 9^{th} *IEEE International Symposium on Industrial Embedded Systems (SIES14)*. Pisa, Italy.

[37] Robert W. Floyd. 1967. Assigning Meaning to Programs. In *Proceedings of Symposium on Applied Mathematics*, Vol. 19. 19–32.

[38] Susanne Graf, Roberto Passerone, and Sophie Quinton. 2014. Contract-Based Reasoning for Component Systems with Rich Interactions. In *Embedded Systems Development: From Functional Models to Implementations*, Alberto L. Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale, and Peter Marwedel (Eds.). Embedded Systems, Vol. 20. Springer New York, Chapter 8, 139–154.

[39] Liangpeng Guo, Qi Zhu, Pierluigi Nuzzo, Roberto Passerone, Alberto L. Sangiovanni-Vincentelli, and Edward A. Lee. 2014. Metronomy: a Function-Architecture Co-simulation Framework for Timing Verification of Cyber-Physical Systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES14)*. ACM, New York, NY, USA, New Delhi, India, Article 24, 10 pages.

[40] Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. 2005. Permissive Interfaces. In *Proceedings of the* 13^{th} *Annual Symposium on Foundations of Software Engineering (FSE05)*. ACM Press, 31–40.

[41] Charles A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (1969), 576–580.

[42] Íñigo Íncer Romeo, Alberto Sangiovanni-Vincentelli, Chung-Wei Lin, and Eunsuk Kang. 2018. Quotient for Assume-guarantee Contracts. In *Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '18)*. Beijing, China, 67–77.

[43] Gabor Karsai, Janos Sztipanovitz, Akos Ledczi, and Ted Bapty. 2003. Model-Integrated Development of Embedded Software. *Proc. IEEE* 91, 1 (January 2003).

[44] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2017. An Extensible Framework for the Exploration of Cyber-Physical System Architectures. In *Proceedings of the* 54^{th} *Design Automation Conference (DAC 2017)*. Austin, TX.

[45] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2018. Optimized selection of wireless network topologies and components via efficient pruning of feasible paths. In *Proceedings of the* 55^{th} *Design Automation Conference (DAC 2018)*. San Francisco, CA.

[46] Leslie Lamport. 1990. win and sin: Predicate Transformers for Concurrency. *ACM Transactions on Programming Languages and Systems* 12, 3 (July 1990), 396–428.

[47] Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 2006. Interface Input/Output Automata. In *14th International Symposium on Formal Methods, FM'06 (Lecture Notes in Computer Science)*, Vol. 4085. Springer, 82–97.

[48] Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 2007. Modal I/O Automata for Interface and Product Line Theories. In *Programming Languages and Systems, 16th European Symposium on Programming, ESOP'07 (Lecture Notes in Computer Science)*, Vol. 4421. Springer, 64–79.

[49] Hoa Thi Thieu Le and Roberto Passerone. 2014. Refinement-based Synthesis of Correct Contract Model Decompositions. In *Proceedings of the* 12^{th} *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE14)*. Lausanne, Switzerland, 134–143.

[50] Hoa Thi Thieu Le, Roberto Passerone, Uli Fahrenberg, and Axel Legay. 2013. A Tag Contract Framework for Heterogeneous Systems. In *Proceedings of the* 12^{th} *International Workshop on Foundations of Coordination Languages and Self Adaptive Systems (FOCLASA13)*. Malaga, Spain, 204–217.

[51] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. 2013. Timed-Automata Based Schedulability Analysis for Distributed Firm Real-Time Systems: a Case Study. *International Journal on Software Tools for Technology Transfer* 15, 3 (June 2013), 211–228.

[52] Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, and Axel Legay. 2013. Tag Machines for Modeling Heterogeneous Systems. In *Proceedings of the* 13^{th} *International Conference on Application of Concurrency to System Design (ACSD13)*. Barcelona, Spain, 186–195.

[53] Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, and Axel Legay. 2016. Contract-based Requirement Modularization via Synthesis of Correct Decompositions. *ACM Transactions on Embedded Computing Systems* 15, 2, Article 33 (2016), 26 pages.

[54] Thi Thieu Hoa Le, Roberto Passerone, Uli Fahrenberg, and Axel Legay. 2016. A Tag Contract Framework for Modeling Heterogeneous Systems. *Science of Computer Programming* 115–116 (2016), 225–246.

[55] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. 2001. Composing domain-specific design environments. *IEEE Computer* 34, 11 (November 2001), 44 –51.

[56] Edward A. Lee and Yuhong Xiong. 2004. A Behavioral Type System and Its Application in Ptolemy II. *Formal Aspects of Computing Journal* 16, 3 (2004), 210 – 237.

[57] Barbara H. Liskov and Jeannette M. Wing. 1994. A Behavioral Notion of Subtyping. *ACM Transactions on Programming Languages and Systems* 16, 6 (November 1994), 1811–1841.

[58] Leonardo Mangeruca, Orlando Ferrante, and Alberto Ferrari. 2013. Formalization and completeness of evolving requirements using contracts. In *Proceedings of the* 8^{th} *IEEE International Symposium on Industrial Embedded Systems (SIES 2013)*. Porto, Portugal, 120–129.
[59] Bertrand Meyer. 1992. Applying "Design by Contract". *IEEE Computer* 25, 10 (October 1992), 40–51.
[60] Ivan Minakov and Roberto Passerone. 2013. PASES: An Energy-Aware Design Space Exploration Framework for Wireless Sensor Networks. *Journal of Systems Architecture* 59, 8 (September 2013), 626–642.
[61] Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, and Sabrina Sicari. 2016. A Comparative Study of Recent Wireless Sensor Network Simulators. *ACM Transactions on Sensor Networks* 12, 3 (2016), 20:1–20:39.
[62] Ivan Minakov, Roberto Passerone, Alessandra Rizzardi, and Sabrina Sicari. 2016. Routing Behavior across WSN Simulators: the AODV Case Study. In *Proceedings of the* 12^{th} *IEEE World Conference on Factory Communication Systems (WFCS 2016)*. Aveiro, Portugal.
[63] Radu Negulescu. 2000. Process Spaces. In *CONCUR 2000 — Concurrency Theory*, Catuscia Palamidessi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 199–213.
[64] Ulrik Nyman. 2008. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. Ph.D. Dissertation. Aalborg University, Department of Computer Science.
[65] OCL 2006. *Object Constraint Language, Version 2.0*. OMG Available Specification formal/06-05-01. Object Management Group.
[66] Roberto Passerone, Jerry R. Burch, and Alberto L. Sangiovanni-Vincentelli. 2004. Conservative Approximations for Heterogeneous Design. In *Proceedings of the Fourth ACM International Conference on Embedded Software (EMSOFT04)*. ACM Press, New York, NY, USA, Pisa, Italy, 155–164.
[67] Roberto Passerone, Jerry R. Burch, and Alberto L. Sangiovanni-Vincentelli. 2007. Refinement Preserving Approximations for the Design and Verification of Heterogeneous Systems. *Formal Methods in System Design* 31, 1 (August 2007), 1–33.
[68] Roberto Passerone, Daniela Cancila, Michele Albano, Sebti Mouelhi, Sandor Plosz, Erkki Jantunen, Anna Ryabokon, Emine Laarouchi, Csaba Hegedűs, and Pal Varga. 2019. A Methodology for the Design of Safety-Compliant and Secure Communication of Autonomous Vehicles. *IEEE Access* (2019). accepted for publication.
[69] Roberto Passerone, Imene Ben Hafaiedh, Susanne Graf, Albert Benveniste, Daniela Cancila, Arnaud Cuccuru, Sébastien Gérard, Francois Terrier, Werner Damm, Alberto Ferrari, Leonardo Mangeruca, Bernhard Josko, Thomas Peikenkamp, and Alberto Sangiovanni-Vincentelli. 2009. Metamodels in Europe: Languages, Tools, and Applications. *IEEE Design and Test of Computers* 26, 3 (May/June 2009), 38–53.
[70] Roberto Passerone, Íñigo Íncer Romeo, and Alberto L. Sangiovanni-Vincentelli. 2019. Coherent Extension, Composition, and Merging Operators in Contract Models for System Design. *ACM Trans. Embed. Comput. Syst.* 18, 5s (Oct. 2019), 86:1–86:23.
[71] Alessandro Pinto, Alvise Bonivento, Alberto L. Sangiovanni-Vincentelli, Roberto Passerone, and Marco Sgroi. 2006. System Level Design Paradigms: Platform-Based Design and Communication Synthesis. *ACM Transactions on Design Automation of Electronic Systems* 11, 3 (July 2006), 537–563.
[72] Alessandro Pinto, Luca P. Carloni, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. 2006. Interchange Formats for Hybrid Systems: Abstract Semantics. In *Hybrid Systems: Computation and Control,* 9^{th} *International Workshop, HSCC06 (Lecture Notes in Computer Science)*, João P. Hespanha and Ashish Tiwari (Eds.), Vol. 3927. Springer, Santa Barbara, CA, USA, 491–506.
[73] Alessandro Pinto, Alberto L. Sangiovanni-Vincentelli, Luca P. Carloni, and Roberto Passerone. 2005. Interchange Formats for Hybrid Systems: Review and Proposal. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC05 (Lecture Notes in Computer Science)*, Manfred Morari and Lothar Thiele (Eds.), Vol. 3414. Springer, Zurich, Switzerland, 526–541.
[74] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. 2009. Modal Interfaces: Unifying Interface Automata And Modal Specifications. In *Proceedings of the Ninth International Conference on Embedded Software (EMSOFT09)*. Grenoble, France, 87–96.
[75] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. 2011. A Modal Interface Theory for Component-based Design. *Fundamenta Informaticae* 108, 1–2 (2011), 119–149.
[76] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, and Roberto Passerone. 2009. Why are Modalities Good for Interface Theories?. In *Proceedings of the Ninth International Conference on Application of Concurrency to System Design (ACSD09)*. Augsburg, Germany, 119–127.
[77] Luca Rizzon and Roberto Passerone. 2016. Cyber/Physical Co-Design in Practice: Case Studies in METROII. In *Proceedings of the* 11^{th} *IEEE International Symposium on Industrial Embedded Systems (SIES16)*. Krakow, Poland.
[78] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. 2012. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *European Journal of Control* 18, 3 (2012), 217–238.
[79] D. Schmidt. 2006. Model-Driven Engineering. *IEEE Computer* (February 2006), 25–31.
[80] M. Törngren, F. Asplund, S. Bensalem, J. McDermid, R. Passerone, H. Pfeifer, A. Sangiovanni-Vincentelli, and B. Schätz. 2016. Characterization, Analysis, and Recommendations for Exploiting the Opportunities of Cyber-Physical Systems. In *Cyber-Physical Systems*, Houbing Song, Danda B. Rawat, Sabina Jeschke, and Christian Brecher (Eds.). Academic Press, Elsevier, Chapter 1, 3–14.
[81] Martin Törngren, Saddek Bensalem, John McDermid, Roberto Passerone, Alberto L. Sangiovanni-Vincentelli, and Bernhard Schätz. 2015. Education and training challenges in the era of Cyber-Physical Systems: beyond traditional engineering. In *Proceedings of the Workshop*

on Embedded and Cyber-Physical Systems Education (WESE 2015). ACM, Amsterdam, The Netherlands, Article 8, 5 pages.

[82] Stavros Tripakis, Ben Lickly, Thomas A. Henzinger, and Edward A. Lee. 2011. A theory of synchronous relational interfaces. *ACM Transactions on Programming Languages and Systems* 33, 4 (July 2011).

[83] Stavros Tripakis, Christos Stergiou, Manfred Broy, and Edward A. Lee. 2013. Error-Completion in Interface Theories. In *Model Checking Software*, Ezio Bartocci and C.R. Ramakrishnan (Eds.). Lecture Notes in Computer Science, Vol. 7976. Springer Berlin Heidelberg, 358–375.

[84] Jos Warmer and Anneke Kleppe. 2003. *The Object Constraint Language: Getting Your Models Ready for MDA* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[85] Elizabeth S. Wolf. 1995. *Hierarchical Models of Synchronous Circuits for Formal Verification and Substitution*. Ph.D. Dissertation. Department of Computer Science, Stanford University.