Refinement-based Synthesis of Correct Contract Model Decompositions

Thi Thieu Hoa Le and Roberto Passerone
Dipartimento di Ingengenria e Scienza dell'Informazione
University of Trento, Italy
email: {hoa.le, roberto.passerone}@unitn.it

***Abstract*—In distributed development of modern systems, contracts play a vital role in ensuring interoperability of components and adherence to specifications. It is therefore often desirable to verify the satisfaction of an overall property represented as a contract, given the satisfaction of smaller properties also represented as contracts. When the verification result is negative, designers must face the issue of refining the sub-properties and components. This is an instance of the classical synthesis problems: "can we construct a model that satisfies some given specification?". In this work, we propose a strategy enabling designers to synthesize or refine a set of contracts so that their composition satisfies a given contract. We develop a generic algebraic method, and show how it can be applied in different contract models to support top-down component-based development of distributed systems.**

I. Introduction

Modern computing systems are increasingly being built by composing components which are developed concurrently by different design teams. In such a paradigm, the distinction between what is constrained on environments, and what must be guaranteed by a system given the constraint satisfaction, reflects the different roles and responsibilities in the system design procedure. Component-based and contract-based design have been shown to be a rigorous and effective approach for designing this kind of concurrent systems [1], [2], [3], [4]. Different components of the same system can be developed by different teams in an independent and concurrent manner provided that their associated contracts can synchronize and satisfy predefined properties. Formally, a contract is a pair of *assumptions* and *guarantees*, which intuitively are properties that must be satisfied by all inputs and outputs of a design, respectively. Such separation between assumptions and guarantees allows an efficient reuse of already-designed components, thereby supporting the distributed development of complex systems effectively.

Components can be formed by a bottom-up composition of simpler predefined components. They can alternatively be formed by a top-down decomposition into sub-components defined by a set of sub-contracts, as long as the composition of the sub-contracts satisfies or refines the contract of the system as a whole. This approach is most appropriate when a design needs to be distributed among several design teams or contractors, since it clearly establishes the responsibilities and decomposes the issue of correctness into smaller local verification subproblems which can be addressed before system integration [5]. The main hurdle with this method is however how to budget the system specification across the different components, by strengthening and weakening their respective assumptions and guarantees, while at the same time achieving overall correctness with respect to the system contract. When this condition is not satisfied, i.e., the sub-contract composition does not refine the overall contract, designers must refine the sub-contract specifications until the system is proved correct. This is an instance of the classical synthesis problems: "Can we construct a model that satisfies some given specification?". In this paper, we deal with the problem of checking if a contract $\mathcal{C}$ can be decomposed into a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ and that of synthesizing the contract set in order to make their composition refine $\mathcal{C}$ when necessary. In particular, we study *decomposing conditions* under which the contract decomposition can be verified, and thereby propose a generic *synthesis strategy* for fixing wrong decompositions. Our conditions and synthesis strategy can be applied to generic contract frameworks equipped with specification operators (e.g., composition, refinement) including popular frameworks like trace-based or modal contract frameworks.

The rest of the paper is organized as follows. In Sect. III, we recall basic notions of contracts and how a generic contract can be derived from a specification theory equipped with composition, quotient, refinement, and conjunction. Based on the decomposition into a set of *two* contracts [2], we propose decomposing conditions for a set of n contracts in Sect. IV and a general strategy for synthesizing those contracts in order to make their composition satisfy a predefined contract in Sect. V. We then demonstrate our strategy for synthesizing trace-based contracts in Sect. VI and modal contracts in Sect. VII. Finally we conclude in Sect. VIII.

II. Related Work

Contracts were first introduced in Bertrand Meyer's design-by-contract method [3], based on ideas by Dijkstra [6], Lamport [7], and others, where systems are viewed as abstract boxes achieving their common goal by verifying specified contracts. Such technique guarantees that methods of a class provide some post-conditions at their termination, as long as the pre-conditions under which they operate are satisfied. De Alfaro and Henzinger subsequently introduced interface automata [4] for documenting components, thereby enabling them to be reused effectively. This formalism establishes a more general notion of contract, where pre-conditions and post-conditions, which originally appeared in the form of predicates, are generalized to behavioral interfaces. The central issues when introducing the formalism of interface automata are compatibility, composition and refinement. Separating assumptions from guarantees, which was somewhat implicit in interface automata, has then been made explicit in the contract framework of the SPEEDS HRC model [8], [9]. A separation between specifying assumptions on expected behaviors and guarantees to achieve them at run time has recently been applied to the handling of synchronization requirements to improve the component-based development of real-time high-integrity systems [10].

The relationship between specifications of component behaviors and contracts is further studied by Bauer et al. [2] where a contract framework can be built on top of any *specification theory* equipped with a composition operator and a refinement relation which satisfy certain properties. Trace-based [8] and modal contract [11] theories are also demonstrated to be instances of such a framework. This formalization enables verifying if a contract can be decomposed into two other contracts by checking if that contract can *dominate* the others. In this work, we take advantage of such a dominating notion and generalize it to a set of n contracts and construct generic decomposing conditions for the contract set.

A number of methods have been reported in the literature for checking contract refinement. Quinton et al. have developed a hierarchical approach that deals with the problem of checking contract refinement by decomposing a large verification task into smaller problems that involve a limited number of assumptions and guarantees, and then relying on compositional methods and circular reasoning to deduce the global result [12]. The tool is part of the BIP framework and uses a model based on modal transition systems. Later, Raclet et al. have developed a theory of contracts based on modal specifications implemented in the InterSMV toolset, dedicated to checking dominance and refinement between contracts [11]. Contract-based specification methods were extended to timed models by David et al. [13], using Timed I/O Automata and constructs for refinement and consistency checking. The theory is implemented on top of the Uppaal-tiga engine for timed games. Benvenuti et al. extend refinement checking to hybrid automata, to account for continuous time components, supported by the Ariadne tool [14]. To improve scalability, Iannapollo et al. propose a library-based approach to refinement checking of contracts expressed in LTL [15]. Our approach is complementary. In fact, while these methods may be effective in determining when refinement holds, they do not provide guidance as to how the decomposition needs to be changed when this is not the case. The verification problem of decomposing a contract into a set of contracts was also studied by Cimatti et al. [16] and was addressed by property-based proof systems with SMT-based model checking techniques, and supported by the OCRA tool. The contract specifications allowed in such systems, however, are trace-based only. Our decomposing conditions can instead deal with generic contract specifications including both trace-based and modal ones.

Assume-Guarantee Reasoning (AGR) has also been applied extensively in declarative compositional reasoning [17] to help prove properties by decomposing them into simpler and more manageable steps. The classical AGR uses assumptions as hypotheses to establish whether a generic property holds. Naturally, this technique can be used in contract models as well, with possibly non-trivial transformation and formalization. In case of unsuccessful termination, AGR can also provide a counterexample showing how the property can be violated. Such a counterexample can then be used to synthesize the model so as to satisfy a given property [18]. However, this synthesis strategy is only applicable for systems with trace-based semantics. Viewing the same assume-guarantee synthesis problem as a game, Chatterjee et al. solve it by finding a winning strategy on the global system state graph, but the method does not guarantee the inclusion of all traces satisfying the specification [19]. The synthesized model was shown to be a subset of that synthesized by counterexample-based synthesis [18]. Unlike these concrete notions of synthesis, ours is more generic since it is not tied to the system semantics. Moreover, while the application of our synthesis strategy to generic contract-based systems is direct and straightforward, the generalization of the previous approaches has not been studied and would require a conversion process from normalized contracts to un-normalized ones.

III. Background: Specification and Contract Theories

A. Specification Theory

For our formalization we follow the notation introduced by Bauer et al. [2] which is built on top of a specification theory equipped with a refinement ($\leq$) and a composition ($\|$) operator. Note that these operators are *meta-theoretical* or uninterpreted operators, meaning that we do not need to know their exact semantics as long as they satisfy certain properties [20]. In particular, monotonicity:

$$(S' \leq S) \wedge (T' \leq T) \Rightarrow (S' \parallel T') \leq (S \parallel T).$$

In addition, composition is commutative and associative while refinement is reflexive and transitive. Two other operators that can be defined on top of composition and refinement are quotient ($/$) and conjunction ($\curlywedge$).

While the refinement operator can relate *concrete* and *abstract* specifications, the composition and quotient, which are dual to each other, can combine specifications to create new ones. Intuitively, the quotient between specifications T and S is a specification R such that its composition with S can concretize or refine T, i.e.

$$S \parallel R \leq T.$$

As there may exist many such specifications, the quotient is defined to be the maximal specification in the refinement order of all such R:

$$((S \parallel (T/S)) \leq T) \wedge ((S \parallel R) \leq T \Rightarrow R \leq T/S)$$

Likewise, the conjunction operator computes the greatest lower bound in the refinement order of the original specification:

$$\begin{gathered}((S \curlywedge T) \leq S) \wedge \\ ((S \curlywedge T) \leq T) \wedge \\ (R \leq S \wedge R \leq T \Rightarrow R \leq (S \curlywedge T))\end{gathered}$$

B. Contract Theory

Assuming the existence of such underlying specification theory, a contract of a component can be defined formally as a pair of specifications, i.e., *assumptions* and *guarantees*:

$$\mathcal{C} = (\mathcal{A}, \mathcal{G}).$$

The specification $\mathcal{A}$ expresses what is constrained on the environments of the component and the specification $\mathcal{G}$ describes what the component can guarantee given the assumption satisfaction. An implementation of the component satisfies its contract whenever it satisfies the contract guarantee, subject to the contract assumption. The contract semantics is therefore defined through the notions of such environments and implementations.

An environment $\mathcal{E}$ satisfies contract $\mathcal{C}$ whenever $\mathcal{E} \leq \mathcal{A}$. Let $[\![\mathcal{C}]\!]_\mathsf{e}$ be the set of environments of $\mathcal{C}$, an implementation $\mathcal{I}$ satisfies contract $\mathcal{C}$ if

$$\forall \mathcal{E} \in [\![\mathcal{C}]\!]_\mathsf{e} : \mathcal{I} \parallel \mathcal{E} \leq \mathcal{G} \parallel \mathcal{E}$$

holds. We denote the set of all possible implementation similarly by $[\![\mathcal{C}]\!]_\mathsf{p}$. Two contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ have identical semantics and are *equivalent* if they possess the same set of environments and implementations, i.e.

$$([\![\mathcal{C}_1]\!]_\mathsf{e} = [\![\mathcal{C}_2]\!]_\mathsf{e}) \wedge ([\![\mathcal{C}_1]\!]_\mathsf{p} = [\![\mathcal{C}_2]\!]_\mathsf{p}).$$

The implementation semantics of a contract, namely its sets of implementations $[\![\mathcal{C}]\!]_\mathsf{p}$ in general depends on both the assumption $\mathcal{A}$ and the guarantee $\mathcal{G}$. Without loss of generality [2], we assume that for every contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$, there exists contract $\mathcal{C}^\mathtt{n} = (\mathcal{A}, \mathcal{G}^\mathtt{n})$ which is equivalent to $\mathcal{C}$ and where the implementation semantics is independent of the assumption presence. This happens when there is a way to incorporate the original assumption and guarantee into the new guarantee $\mathcal{G}^\mathtt{n}$. We call $\mathcal{C}^\mathtt{n}$ the normalized form of $\mathcal{C}$ and derive $\mathcal{G}^\mathtt{n}$ using the normalization operator $\triangleright$ which can be defined on top of the basic operators $\leq$, $\parallel$, $/$, $\curlywedge$:

$$\mathcal{G}^\mathtt{n} = \mathcal{G} \triangleright \mathcal{A}.$$

Definition 1. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ is in normalized form if and only if:

$$\mathcal{I} \in [\![\mathcal{C}]\!]_\mathsf{p} \Leftrightarrow \mathcal{I} \leq \mathcal{G}.$$

A refinement relation between contracts can then be established based on that between their environment sets and implementation sets.

Definition 2. Contract $\mathcal{C}$ is said to refine $\mathcal{C}'$, written $\mathcal{C} \prec \mathcal{C}'$, when it can accept more environments and fewer implementations than contract $\mathcal{C}'$:

$$[\![\mathcal{C}']\!]_\mathsf{e} \subseteq [\![\mathcal{C}]\!]_\mathsf{e} \wedge [\![\mathcal{C}]\!]_\mathsf{p} \subseteq [\![\mathcal{C}']\!]_\mathsf{p}.$$

IV. Contract Composition and Decomposition

Composing contracts is formalized so that the compositionality between their implementations can be respected, i.e., composing such implementations results in an implementation of the composite contract. In addition, every environment of the composite contract should be able to work with any implementation of an individual contract in a way that their composition does not violate the other contract assumption. In fact, there exists a class of contracts, including the composite contract, able to provide such desirable consequences. These are referred to as *dominating* contracts [2] and the composite contract is the least in the refinement order of all dominating contracts, as we shall see in Section IV-A.

This notion of dominance thus enables the compositionality of the implementation relation, an important principle in reusing components and decomposing systems into existing components. Before studying contract decomposition (Section IV-B), we first generalize the notion of dominance and composition from two contracts [2] to a set of n contracts.

A. Contract Composition

Definition 3. A contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ *dominates* the contract set $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ if:

i) $\forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_\mathsf{p}, \ldots, \forall \mathcal{I}_n \in [\![\mathcal{C}_n]\!]_\mathsf{p} : \underset{1 \leq i \leq n}{\parallel} \mathcal{I}_i \in [\![\mathcal{C}]\!]_\mathsf{p}$,

ii) $\forall \mathcal{E} \in [\![\mathcal{C}]\!]_\mathsf{e}, \forall \mathcal{I}_1 \in [\![\mathcal{C}_1]\!]_\mathsf{p}, \ldots, \forall \mathcal{I}_n \in [\![\mathcal{C}_n]\!]_\mathsf{p}, \forall 1 \leq i \leq n :$

$$\mathcal{E} \parallel \underset{1 \leq j \neq i \leq n}{\Big\|} \mathcal{I}_j \leq \mathcal{A}_i.$$

The following theorem reduces checking the two conditions in Definition 3 to checking simpler formulas.

Theorem 1. *Condition* (i) *is equivalent to*

$$\underset{1 \leq i \leq n}{\Big\|} \mathcal{G}_i^\mathtt{n} \in [\![\mathcal{C}]\!]_\mathsf{p}.$$

Condition (ii) *is equivalent to*

$$\forall 1 \leq i \leq n : \mathcal{A} \parallel \underset{1 \leq j \neq i \leq n}{\Big\|} \mathcal{G}_j^\mathtt{n} \leq \mathcal{A}_i.$$

Proof:

i) $\Rightarrow$: Consider $\mathcal{I}_i = \mathcal{G}_i^\mathtt{n}$.
$\Leftarrow$: $\mathcal{I}_i \leq \mathcal{G}_i^\mathtt{n}$ implies

$$\forall \mathcal{E} \in [\![\mathcal{C}]\!]_\mathsf{e} : (\mathcal{E} \parallel \underset{1 \leq i \leq n}{\Big\|} \mathcal{I}_i) \leq (\mathcal{E} \parallel \underset{1 \leq i \leq n}{\Big\|} \mathcal{G}_i^\mathtt{n})$$

which in turn implies

$$\underset{1 \leq i \leq n}{\Big\|} \mathcal{I}_i \in [\![\mathcal{C}]\!]_\mathsf{p}.$$

ii) $\Rightarrow$: Consider $\mathcal{E} = \mathcal{A}, \mathcal{I}_j = \mathcal{G}_j^\mathtt{n}$.
$\Leftarrow$: $(\mathcal{E} \parallel \underset{1 \leq j \neq i \leq n}{\parallel} \mathcal{I}_j) \leq (\mathcal{A} \parallel \underset{1 \leq j \neq i \leq n}{\parallel} \mathcal{G}_j^\mathtt{n}) \leq \mathcal{A}_i$.

■

The composition of a set of contracts can then be defined as follows.

Definition 4. The composition of a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, written $\underset{1 \leq i \leq n}{\bigodot} \mathcal{C}_i$, is the contract

$$\mathcal{C} = (\mathcal{A}, \mathcal{G}) = (\underset{1 \leq i \leq n}{\curlywedge} (\mathcal{A}_i / \underset{1 \leq k \neq i \leq n}{\Big\|} \mathcal{G}_k^\mathtt{n}), \underset{1 \leq j \leq n}{\Big\|} \mathcal{G}_j^\mathtt{n}).$$

Let contracts $\mathcal{C}_i, \mathcal{C}_i'$ be such that $\mathcal{C}_i' \prec \mathcal{C}_i$. The following theorem generalizes the following important results which were established for $n = 2$ [2]:

- The composition of a set of contracts dominates the individual contracts (Theorem 2 (i)) and is the *least*, in

the refinement order, of all contracts dominating them (Theorem 2 (ii)).
- Dominance is preserved under the refinement operation of contracts (Theorem 2 (iiia)).
- Contract refinement is preserved under contract composition (Theorem 2 (iiib)).

As a result, we can derive a generic contract theory with an $n-$ary structural composition for contracts, lifting the compositional design to a set of n components.

Theorem 2. *Let $\mathcal{C}$ be the composition of $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, then:*

i) $\mathcal{C}$ dominates the contract set $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$.

ii) $\forall \mathcal{C}' : \mathcal{C}'$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\} \Leftrightarrow \mathcal{C} \prec \mathcal{C}'$.

iii) If $\mathcal{C}'$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ then:

a) it dominates also $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}$ where $\mathcal{C}'_i \prec \mathcal{C}_i$,

b) $(\bigodot_{1\leq i\leq n} \mathcal{C}'_i) \prec (\bigodot_{1\leq i\leq n} \mathcal{C}_i)$.

Proof: Let $\mathcal{A}^{/}_h$ be defined as follows:

$$\mathcal{A}^{/}_h \overset{\text{def}}{=} \mathcal{A}_h / \mathop{\Big|\Big|}_{1\leq k\neq h\leq n} \mathcal{G}^{\mathsf{n}}_k,$$

then $\mathcal{A} = \bigcurlywedge_{1\leq h\leq n} \mathcal{A}^{/}_h$ and it follows that $\mathcal{A} \leq \mathcal{A}^{/}_h$.

i) $\mathcal{C}$ dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ because:

a) $\mathop{\Big|\Big|}_{1\leq i\leq n} \mathcal{I}_i \in [\![\mathcal{C}]\!]_{\mathsf{p}}$, by Theorem 1 and $\mathcal{G} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$.

b) By Theorem 1 and by the quotient property:

$$\begin{aligned}\mathcal{A} \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j &\leq \mathcal{A}^{/}_i \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j \\ &\leq (\mathcal{A}_i / \mathop{\Big|\Big|}_{1\leq k\neq i\leq n} \mathcal{G}^{\mathsf{n}}_k) \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j \\ &\leq \mathcal{A}_i.\end{aligned}$$

ii)
- $\Rightarrow$: By the dominance of $\mathcal{C}'$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ and by Theorem 1, we have:

$$\begin{aligned}\mathcal{A}' \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j \leq \mathcal{A}_i &\Rightarrow \mathcal{A}' \leq \mathcal{A}_i / \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j \\ &\Rightarrow \mathcal{A}' \leq \bigcurlywedge_{1\leq i\leq n} (\mathcal{A}_i / \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j) \\ &\Rightarrow \mathcal{A}' \leq \bigcurlywedge_{1\leq i\leq n} \mathcal{A}^{/}_i.\end{aligned}$$

This means $\mathcal{A}' \leq \mathcal{A}$ and implies $[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}}$ which in turn implies:

$$\forall \mathcal{E}' \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{E}' \in [\![\mathcal{C}]\!]_{\mathsf{e}}$$

We also have $\mathcal{I} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$ which means:

$$\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \leq \mathcal{G} \parallel \mathcal{E}$$

By the dominance of $\mathcal{C}'$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, the following is true:

$$\mathcal{G} \in [\![\mathcal{C}']\!]_{\mathsf{p}} \Rightarrow \forall \mathcal{E}' \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{G} \parallel \mathcal{E}' \leq \mathcal{G}' \parallel \mathcal{E}'$$

Combining all of the above together, we have:

$$\begin{aligned}&\forall \mathcal{E} \in [\![\mathcal{C}]\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E} \leq \mathcal{G} \parallel \mathcal{E} \\ \Rightarrow\ &\forall \mathcal{E}' \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E}' \leq \mathcal{G} \parallel \mathcal{E}' \\ \Rightarrow\ &\forall \mathcal{E}' \in [\![\mathcal{C}']\!]_{\mathsf{e}} : \mathcal{I} \parallel \mathcal{E}' \leq \mathcal{G}' \parallel \mathcal{E}' \\ \Rightarrow\ &\mathcal{I} \in [\![\mathcal{C}']\!]_{\mathsf{p}}.\end{aligned}$$

This implies $[\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}$ and therefore $\mathcal{C} \prec \mathcal{C}'$.
- $\Leftarrow$: The refinement relation $\mathcal{C} \prec \mathcal{C}'$ means

$$([\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}) \wedge ([\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}}).$$

Since $\mathcal{G} \in [\![\mathcal{C}]\!]_{\mathsf{p}}$ and $[\![\mathcal{C}]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}']\!]_{\mathsf{p}}$, we then have:

$$\mathop{\Big|\Big|}_{1\leq i\leq n} \mathcal{G}^{\mathsf{n}}_i \in [\![\mathcal{C}']\!]_{\mathsf{p}}.$$

In addition,

$$\begin{aligned}[\![\mathcal{C}']\!]_{\mathsf{e}} \subseteq [\![\mathcal{C}]\!]_{\mathsf{e}} &\Rightarrow \mathcal{A}' \leq \mathcal{A} \\ &\Rightarrow \mathcal{A}' \leq \mathcal{A}^{/}_i \\ &\Rightarrow \mathcal{A}' \leq \mathcal{A}_i / \mathop{\Big|\Big|}_{1\leq k\neq i\leq n} \mathcal{G}^{\mathsf{n}}_k \\ &\Rightarrow (\mathcal{A}' \parallel \mathop{\Big|\Big|}_{1\leq k\neq i\leq n} \mathcal{G}^{\mathsf{n}}_k) \leq \mathcal{A}_i.\end{aligned}$$

By Theorem 1, $\mathcal{C}'$ then dominates $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$.

iii) a) First,

$$\begin{aligned}\mathcal{C}'_i \prec \mathcal{C}_i &\Rightarrow [\![\mathcal{C}'_i]\!]_{\mathsf{p}} \subseteq [\![\mathcal{C}_i]\!]_{\mathsf{p}} \\ &\Rightarrow \mathcal{I}'_i \in [\![\mathcal{C}_i]\!]_{\mathsf{p}} \\ &\Rightarrow \mathop{\Big|\Big|}_{1\leq i\leq n} \mathcal{I}'_i \in [\![\mathcal{C}']\!]_{\mathsf{p}}\end{aligned}$$

(the last implication is because of the dominance of $\mathcal{C}'$ over $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$). Second,

$$\begin{aligned}\mathcal{C}'_i \prec \mathcal{C}_i &\Rightarrow \mathcal{G}'^{\mathsf{n}}_i \leq \mathcal{G}^{\mathsf{n}}_i \\ &\Rightarrow \mathcal{A}' \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}'^{\mathsf{n}}_j \leq \mathcal{A}' \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}^{\mathsf{n}}_j \\ &\Rightarrow \mathcal{A}' \parallel \mathop{\Big|\Big|}_{1\leq j\neq i\leq n} \mathcal{G}'^{\mathsf{n}}_j \leq \mathcal{A}_i \leq \mathcal{A}'_i.\end{aligned}$$

(the last implication is because of $\mathcal{C}'_i \prec \mathcal{C}_i$).
By Theorem 1, $\mathcal{C}'$ thus dominates $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}$.

b) A direct consequence of items (i), (ii), (iiia) of Theorem 2.

■

B. Contract Decomposition

As a direct consequence of Theorem 2 (ii), a contract $\mathcal{C}$ refined by the composition of a set of contracts $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ will dominate that contract set and provide desirable compositional consequences formalized in items (i) and (ii) of Definition 3. This contract set is then considered to be a *decomposition* of $\mathcal{C}$, allowing the components associated with the contract set or their refinements to be plugged into a system satisfying contract $\mathcal{C}$ without breaking the contract satisfaction.

Verifying if $\mathcal{C}$ can be decomposed into $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is therefore equivalent to checking the dominance of $\mathcal{C}$ over

$\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ which, by Theorem 1, corresponds to the two decomposing conditions (**DC**s):

$$\textbf{DC-1)} \; \mathop{\Big\|}_{1\le i\le n} \mathcal{G}_i^{\mathsf{n}} \in [\![\mathcal{C}]\!]_{\mathsf{p}}, \text{ or equivalently } \mathop{\Big\|}_{1\le i\le n} \mathcal{G}_i^{\mathsf{n}} \le \mathcal{G}^{\mathsf{n}}$$

$$\textbf{DC-2)} \; \forall 1 \le i \le n : \mathcal{A} \parallel \mathop{\Big\|}_{1\le j\ne i\le n} \mathcal{G}_j^{\mathsf{n}} \le \mathcal{A}_i.$$

Moreover, our extension on the dominance notion is more generic than that of Cimatti et al. [16] and can support the construction of property-based proof systems such as that proposed by the same authors. In fact, we built our system in a generic way using a set of meta-theoretical operators including composition, refinement, quotient and conjunction. Our extension can therefore be applied to build proof systems of different contract frameworks where these operators are explicitly instantiated. For example, trace-based contract system development [16] can be derived by instantiating the composition and refinement between specifications as the intersection and set inclusion as follows:

$$\text{i)} \; \bigcap_{1\le i\le n} \mathcal{G}_i^{\mathsf{n}} \in [\![\mathcal{C}]\!]_{\mathsf{p}}, \text{ or equivalently } \bigcap_{1\le i\le n} \mathcal{G}_i^{\mathsf{n}} \subseteq \mathcal{G}^{\mathsf{n}}$$
$$\text{ii)} \; \forall 1 \le i \le n : \mathcal{A} \cap \bigcap_{1\le j\ne i\le n} \mathcal{G}_j^{\mathsf{n}} \subseteq \mathcal{A}_i.$$

Likewise, modal contract system development can be based on the modal alternating refinement $\le_m$ and the modal composition $\|_m$ on shared actions [2].

V. Contract Synthesis

When a set of contracts does not satisfy the decomposing conditions established in Section IV-B, we must adjust the specification of some of them. We propose a synthesis strategy based on the following condition which says that the conjunction operator can be distributed over the normalizing operator $\rhd$ as follows:

$$(\mathcal{G} \rhd \mathcal{A}) \curlywedge X = (\mathcal{G} \curlywedge X) \rhd (\mathcal{A} \rhd X) \qquad (1)$$

Although this condition poses certain limitations on contract systems, it is a desirable property because it shows that the semantics of a model is invariant when commuting (appropriately) normalization $\rhd$ and conjunction $\curlywedge$. Better flexibility in the design process can also be gained when these operators are commutative. Since conjunction and normalization amount to strengthening and weakening operations respectively, strengthening X causes a semantic reduction in the two sides of equation (1). Thus, when this property does not hold, we can keep strengthening X until we reach a fixed point in semantic equivalence as we shall see later in Section VII.

Contract synthesis consists of finding suitable refinements for the individual contracts. Our synthesis strategy is based on strengthening the normalized guarantees, which can be reduced to strengthening the un-normalized guarantees and weakening the corresponding assumptions. Because such operations either strengthen the left sides or weaken the right sides of the decomposing conditions, their refinement relation are either maintained or changed from false to true.

A. Synthesis of Decomposing Conditions

To satisfy **DC-1**, we randomly select a guarantee $\mathcal{G}_k^{\mathsf{n}}$ to be strengthened where $\mathcal{G}_k^{\mathsf{n}}$ can be any of the guarantees $\mathcal{G}_i^{\mathsf{n}}$ composing **DC-1**. By taking advantage of the quotient, we can find the least specification

$$X = \mathcal{G}^{\mathsf{n}} / (\mathop{\Big\|}_{1\le i\ne k\le n} \mathcal{G}_i^{\mathsf{n}})$$

which ensures the satisfaction of **DC-1**. The newly strengthened normalized guarantee $\bar{\mathcal{G}_k^{\mathsf{n}}}$ is then:

$$\bar{\mathcal{G}_k^{\mathsf{n}}} = \mathcal{G}_k^{\mathsf{n}} \curlywedge X = (\mathcal{G}_k \rhd \mathcal{A}_k) \curlywedge X = (\mathcal{G}_k \curlywedge X) \rhd (\mathcal{A}_k \rhd X) \qquad (2)$$

Since conjunction and normalization amount to strengthening and weakening operations respectively, the above equation shows that strengthening a normalized guarantee amounts to strengthening its un-normalized version and weakening its coupled assumption. Overall, it amounts to refining the contract $\mathcal{C}_k$. It is also important to notice that strengthening $\mathcal{G}_k^{\mathsf{n}}$ as above either maintains the refining property established in **DC-2** or may change it from false to true, but not vice-versa because:

$$\begin{aligned} &\mathcal{A}_k \le \mathcal{A}_k \rhd X, \text{ for } i = k,\\ \mathcal{A} \parallel \bar{\mathcal{G}_k^{\mathsf{n}}} \parallel \mathop{\Big\|}_{1\le j\ne k, i\le n} \mathcal{G}_j^{\mathsf{n}} \le \mathcal{A} \parallel &\mathop{\Big\|}_{1\le j\ne i\le n} \mathcal{G}_j^{\mathsf{n}}, \text{ for } i \ne k. \end{aligned}$$

In order to satisfy the i-th clause of **DC-2**, we randomly select a guarantee $\mathcal{G}_{k_i}^{\mathsf{n}}$ to be strengthened where $\mathcal{G}_{k_i}^{\mathsf{n}}$ can be any of the guarantees $\mathcal{G}_j^{\mathsf{n}}$ composing the i-th clause of **DC-2** and $k_i \ne i$. We then can find the least specification

$$Y_i = \mathcal{A}_i / (\mathcal{A} \parallel \mathop{\Big\|}_{1\le j\ne i, k_i\le n} \mathcal{G}_j^{\mathsf{n}})$$

which ensures the satisfaction of the i-th clause. As done for condition 1, $\mathcal{G}_{k_i}^{\mathsf{n}}$ is strengthened to $\bar{\mathcal{G}_{k_i}^{\mathsf{n}}} \overset{\text{def}}{=} \mathcal{G}_{k_i}^{\mathsf{n}} \curlywedge Y_i$:

$$\bar{\mathcal{G}_{k_i}^{\mathsf{n}}} = \mathcal{G}_{k_i}^{\mathsf{n}} \curlywedge Y_i = (\mathcal{G}_{k_i} \rhd \mathcal{A}_{k_i}) \curlywedge Y_i = (\mathcal{G}_{k_i} \curlywedge Y_i) \rhd (\mathcal{A}_{k_i} \rhd Y_i) \qquad (3)$$

B. Synthesis Strategy

Based on equation (1) and the above analysis, we propose a strategy for synthesizing $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ in order to make it a decomposition of $\mathcal{C}$ as follows:

1) If **DC-1** is not satisfied, randomly select a contract $\mathcal{C}_k$ to be refined and apply (2).
2) While **DC-2** is not satisfied:
   a) Let i be the index of an unsatisfied clause, randomly select contract $\mathcal{C}_{k_i}$ to be refined and apply (3).
   b) Repeat step (2a) until **DC-2** is satisfied.

We next demonstrate our strategy in synthesizing trace-based and modal contract sets.

VI. Trace-based Contract Synthesis

In trace-based contract systems, assumptions and guarantees are considered as sets of traces (or behaviors) defined over a set of system ports (or variables). Every trace assign a history of values to ports. In this setting, refinement is defined as set inclusion, composition and conjunction is intersection and normalization, which is identical to quotient, is defined as

$$\mathcal{G} \rhd \mathcal{A} = \mathcal{G}/\mathcal{A} = \mathcal{G} \cup \neg\mathcal{A},$$

it is easy to verify that equation (1) is satisfied:

$$(\mathcal{G} \cup \neg\mathcal{A}) \cap X = (\mathcal{G} \cap X) \cup \neg(\mathcal{A} \cup \neg X).$$

Therefore, we can apply the synthesis strategy proposed above directly. It is also interesting to notice that for trace-based models, to satisfy the i-th clause of **DC-2**, an alternative is to weaken $\mathcal{A}_i$ to $\bar{\mathcal{A}}_i$:

$$\bar{\mathcal{A}}_i \stackrel{\text{def}}{=} \mathcal{A}_i \cup Z_i$$

where $\mathcal{A} \cap \bigcap_{1 \le j \ne i \le n} \mathcal{G}_j^{\mathrm{n}} \subseteq Z_i$. This operation has a nice consequence in strengthening the corresponding normalized guarantee which is

$$\bar{\mathcal{G}_i^{\mathrm{n}}} = \mathcal{G}_i \cup \neg(\mathcal{A}_i \cup Z_i) = \mathcal{G}_i \cup (\neg\mathcal{A}_i \cap \neg Z_i)$$

since $(\neg\mathcal{A}_i \cap \neg Z_i) \subseteq \neg\mathcal{A}_i \Rightarrow \bar{\mathcal{G}_i^{\mathrm{n}}} \subseteq \mathcal{G}_i^{\mathrm{n}}$.

Example 1. We consider a variant of the contract model of the Brake System Control Unit (BSCU) described in [21] and shown in Figure 1(a). The BSCU takes as inputs the positions of the two brake pedals $\texttt{Pedal\_Pos}_1$ and $\texttt{Pedal\_Pos}_2$ and outputs two control signals Valid and CMD_AS to control the braking process of a wheel-brake system.

The BSCU component is further decomposed into a Select_Switch and two smaller control units: a primary $\texttt{BSCU}_1$ and a backup $\texttt{BSCU}_2$. When $\texttt{BSCU}_1$ fails, the Select_Switch puts the backup signal from $\texttt{BSCU}_2$ through. The signal failure in a control unit $\texttt{BSCU}_i$ is indicated by its signal $\texttt{Valid}_i$ going down and is caused by a basic fault which is either a monitor fault $\texttt{fault\_Monitor}_i$ or a command fault $\texttt{fault\_Command}_i$ with $i \in \{1, 2\}$.

A safety requirement on the BSCU is to ensure that $\texttt{Valid}_1 \lor \texttt{Valid}_2$ is always true when at most one of the basic faults $\texttt{fault\_Monitor}_i$ or $\texttt{fault\_Command}_i$ can occur [16]. This is specified as contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ in Fig. 1(b). The safety contracts specification [16] on $\texttt{BSCU}_i$ make no assumptions and guarantees that signal $\texttt{Valid}_i$ remains true when neither of its basic faults occurs. In this example, we strengthen the assumption of the original safety contracts on $\texttt{BSCU}_i$ and present them as contract $\mathcal{C}_i = (\mathcal{A}_i, \mathcal{G}_i)$ in Fig. 1(b)

The contracts can be specified in symbolic logic [16] where sets of traces are represented by logical formulas. Thus checking the two **DC**s amounts to checking the following formulas in symbolic logic:

i) $\bigwedge_{1 \le i \le n} \mathcal{G}_i^{\mathrm{n}} \Rightarrow \mathcal{G}^{\mathrm{n}}$

ii) $\forall 1 \le i \le n : \mathcal{A} \land \bigwedge_{1 \le j \ne i \le n} \mathcal{G}_j^{\mathrm{n}} \Rightarrow \mathcal{A}_i$.

where $\mathcal{G}^{\mathrm{n}} = \mathcal{G} \lor \neg\mathcal{A}$ and $\mathcal{G}_i^{\mathrm{n}} = \mathcal{G}_i \lor \neg\mathcal{A}_i$.

In order to reuse the contract specification of the subcomponents $\texttt{BSCU}_1$ and $\texttt{BSCU}_2$, we verify if $\mathcal{C}$ can be decomposed into $\mathcal{C}_1$ and $\mathcal{C}_2$ which amounts to verifying the satisfaction of the two **DC**s. While $\mathcal{C}$ can be decomposed into the subcomponents' original contracts [16], it cannot be decomposed into $\mathcal{C}_1$ and $\mathcal{C}_2$ without refining $\mathcal{C}_i$ as we shall show below.

It is obvious that the contracts $\mathcal{C}_i$ are in normal form, thus $\mathcal{G}_i^{\mathrm{n}} \equiv \mathcal{G}_i$. We observe that **DC-1** is satisfied because $\mathcal{G}_1 \land \mathcal{G}_2 \Rightarrow \mathcal{G}^{\mathrm{n}}$ is true. However **DC-2** is not satisfied because $\mathcal{A} \land \mathcal{G}_1 \Rightarrow \mathcal{A}_2$ is not true. Applying step (2a) of our synthesis strategy, we refine $\mathcal{C}_1$ w.r.t. Y_2 into $\mathcal{C}_1' = (\mathcal{A}_1', \mathcal{G}_1')$ where:

$$\begin{aligned} Y_2 &= (\mathcal{A}_2/\mathcal{A}) = (\mathcal{A}_2 \cup \neg\mathcal{A}) = (\mathcal{A} \Rightarrow \mathcal{A}_2), \\ \mathcal{A}_1' &= (\mathcal{A}_1 \rhd Y_2) = (\mathcal{A}_1/Y_2) = (\mathcal{A}_1 \cup \neg Y_2), \\ \mathcal{G}_1' &= (\mathcal{G}_1 \curlywedge Y_2) = (\mathcal{G}_1 \cap Y_2). \end{aligned}$$

DC-2 is still not satisfied after the first synthesis because $\mathcal{A} \land \mathcal{G}_2 \Rightarrow \mathcal{A}_1'$ is not true. Applying step (2a) once again, we refine $\mathcal{C}_2$ w.r.t. Y_1 into $\mathcal{C}_2' = (\mathcal{A}_2', \mathcal{G}_2')$ where:

$$\begin{aligned} Y_1 &= (\mathcal{A}_1'/\mathcal{A}) = (\mathcal{A}_1 \cup \neg Y_2) \cup \neg\mathcal{A} = ((\mathcal{A} \land \mathcal{A}_2) \Rightarrow \mathcal{A}_1), \\ \mathcal{A}_2' &= (\mathcal{A}_2 \rhd Y_1) = (\mathcal{A}_2/Y_1) = (\mathcal{A}_2 \cup \neg Y_1), \\ \mathcal{G}_2' &= (\mathcal{G}_2 \curlywedge Y_1) = (\mathcal{G}_2 \cap Y_1). \end{aligned}$$

Alternatively, we can weaken $\mathcal{A}_i$ w.r.t. any Z_i such that

$$(\mathcal{A} \land \mathcal{G}_{3-i}) \Rightarrow Z_i$$

is correct. The simplest option could be $Z_i = \texttt{TRUE}$ and this derives the original safety contracts [16]. Our approach therefore provides a wider set of options which allows designers to explore the refinement space.

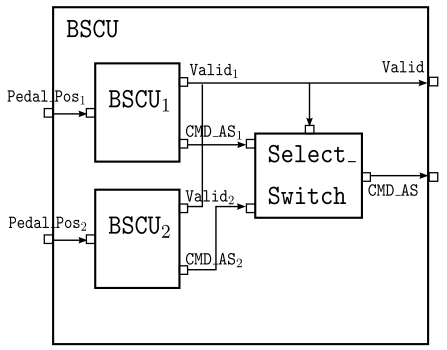
VII. Modal Contract Synthesis

Modal contracts are defined over modal transition systems (MTS) where transitions are annotated with action labels and with *may* or *must* modalities modeling behaviors which can be (optionally) or must be (compulsorily) implemented respectively. Formally, an MTS is a tuple $M = (S, s_0, \Sigma, \dashrightarrow, \rightarrow)$ where S is the set of states, $s_0 \in S$ is the initial state, Σ is the set of actions and $\dashrightarrow, \rightarrow \subseteq S \times \Sigma \times S$ are the *may*,*must* transition relation respectively such that $\rightarrow \subseteq \dashrightarrow$ [2].

For the sake of comprehension, we use our notations with $m-$subscripts when referring to modal operators. The modal operators for combining modal transitions are described in Table I where ⓤ denotes a new state in which there is a looping *may* transition for every action. This state is referred to as the *universal* state. Let *may*(s_i) and *must*(s_i) denote the set of *may* actions and *must* actions allowed respectively at state s_i. State s_i is *consistent* when *must*$(s_i) \subseteq$ *may*(s_i). Combining modal systems using operators presented in Table I may introduce inconsistent states. A pruning procedure [11] is therefore required to remedy such a problem. For the sake of completeness, we shall recall briefly this procedure.

Let $M = (S, s_0, \Sigma, \dashrightarrow, \rightarrow)$ be the newly combined system containing inconsistent states. Let $M_0 = (S, s_0, \Sigma, \dashrightarrow_0, \rightarrow_0)$ be a copy of M, i.e. $\dashrightarrow_0 \equiv \dashrightarrow$ and $\rightarrow_0 \equiv \rightarrow$ and let $k = 0$, we obtain the pruning of M through the following steps:

1) Let $M_{k+1} = (S, s_0, \Sigma, \dashrightarrow_{k+1}, \rightarrow_{k+1})$ be a copy of M_k.
2) For each run $r = s_0 \overset{\sigma_0}{\dashrightarrow} s_1 \ldots \overset{\sigma_{n-1}}{\dashrightarrow} s_n$ from the initial state s_0 to state s_n in M_k where $\mathit{must}_k(s_n) \nsubseteq \mathit{may}_k(s_n)$:
    i) set $\mathit{may}_{k+1}(s_n) = \Sigma$ and $\mathit{must}_{k+1}(s_n) = \emptyset$,
    ii) set $\mathit{may}_{k+1}(s_{n-1}) = \mathit{may}_k(s_{n-1}) \setminus \{\sigma_{n-1}\}$.



(a) High-level view of BSCU

$$\begin{aligned}
\mathcal{A} &\overset{\text{def}}{=} \texttt{always}\ (\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Monitor}_2) \vee \\
&\quad \texttt{always}\ (\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Command}_2) \vee \\
&\quad \texttt{always}\ (\neg\texttt{fault\_Monitor}_1 \wedge \neg\texttt{fault\_Command}_2 \wedge \neg\texttt{fault\_Monitor}_2) \vee \\
&\quad \texttt{always}\ (\neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Command}_1 \wedge \neg\texttt{fault\_Monitor}_2) \\
\mathcal{G} &\overset{\text{def}}{=} \texttt{always}\ (\texttt{Valid}_1 \vee \texttt{Valid}_2) \\
\mathcal{A}_i &\overset{\text{def}}{=} \texttt{always}\ (\neg\texttt{fault\_Monitor}_i) \wedge \texttt{always}\ (\neg\texttt{fault\_Command}_i) \\
\mathcal{G}_i &\overset{\text{def}}{=} \texttt{always}\ (\neg\texttt{fault\_Monitor}_i) \wedge \texttt{always}\ (\neg\texttt{fault\_Command}_i) \\
&\qquad \Longrightarrow \texttt{always}\ (\texttt{Valid}_i)
\end{aligned}$$

(b) Contract specification of BSCU and BSCU$_i$

Fig. 1: Structure and contract models of BSCU

TABLE I: Rules for combining modal specifications S_1 and S_2 using modal operators $\rhd_m, \|_m, /_m, \curlywedge_m$

$S_1 \rhd_m S_2$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\not\dashrightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \textcircled{u}$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \textcircled{u}$
$s_1 \overset{\alpha}{\not\dashrightarrow}$		$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \textcircled{u}$

$S_1 \|_m S_2$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$

$S_1 /_m S_2$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\not\dashrightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	(s_1,s_2) is inconsistent	(s_1,s_2) is inconsistent
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \textcircled{u}$
$s_1 \overset{\alpha}{\not\dashrightarrow}$			$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \textcircled{u}$

$S_1 \curlywedge_m S_2$	$s_2 \overset{\alpha}{\rightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \overset{\alpha}{\not\dashrightarrow}$
$s_1 \overset{\alpha}{\rightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	(s_1,s_2) is inconsistent
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\rightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	
$s_1 \overset{\alpha}{\not\dashrightarrow}$	(s_1,s_2) is inconsistent		

3) Set $k = k+1$. If M_k still contains inconsistent states, repeat the above steps. Otherwise the procedure terminates.

The modal refinement is defined as follows [2]. An MTS $M_1 = (S_1, s_{01}, \Sigma_1, \dashrightarrow_1, \rightarrow_1)$ refines another MTS $M_2 = (S_2, s_{02}, \Sigma_2, \dashrightarrow_2, \rightarrow_2)$, written $M_1 \leq_m M_2$, if there exists a relation $R \subseteq S_1 \times S_2$ such that $(s_{01}, s_{02}) \in R$ and for all $(s_1, s_2) \in R$ and $\alpha \in \Sigma$:

$$\begin{aligned}
&((s_1,\alpha,s_1') \in \dashrightarrow_1 \Rightarrow \exists (s_2,\alpha,s_2') \in \dashrightarrow_2 \colon (s_1',s_2') \in R) \wedge \\
&\quad ((s_2,\alpha,s_2') \in \rightarrow_2 \Rightarrow \exists (s_1,\alpha,s_1') \in \rightarrow_1 \colon (s_1',s_2') \in R).
\end{aligned}$$

Consider a simple modal contract $\mathcal{C} = (\mathcal{A}, \mathcal{G})$ specified in Fig. 2(a) and Fig. 2(b) and a specification X in Fig. 2(d) where the initial states are marked by bold circles. Equation (1) is shown to be violated as demonstrated in Fig. 2(g) and Fig. 2(h). The reason is that normalization may introduce a universal state with a looping *may* transition for every action. Whereas, during conjunction, such universal state could be pruned away. To avoid such inconsistency, $\mathcal{A} \rhd_m X$ should contain all *may* transitions appearing in X and we observe that it can be obtained by tightening X to

$$\bar{X} \overset{\text{def}}{=} X \curlywedge_m \mathcal{A}$$

as shown in Fig. 2(l) and Fig. 2(m).

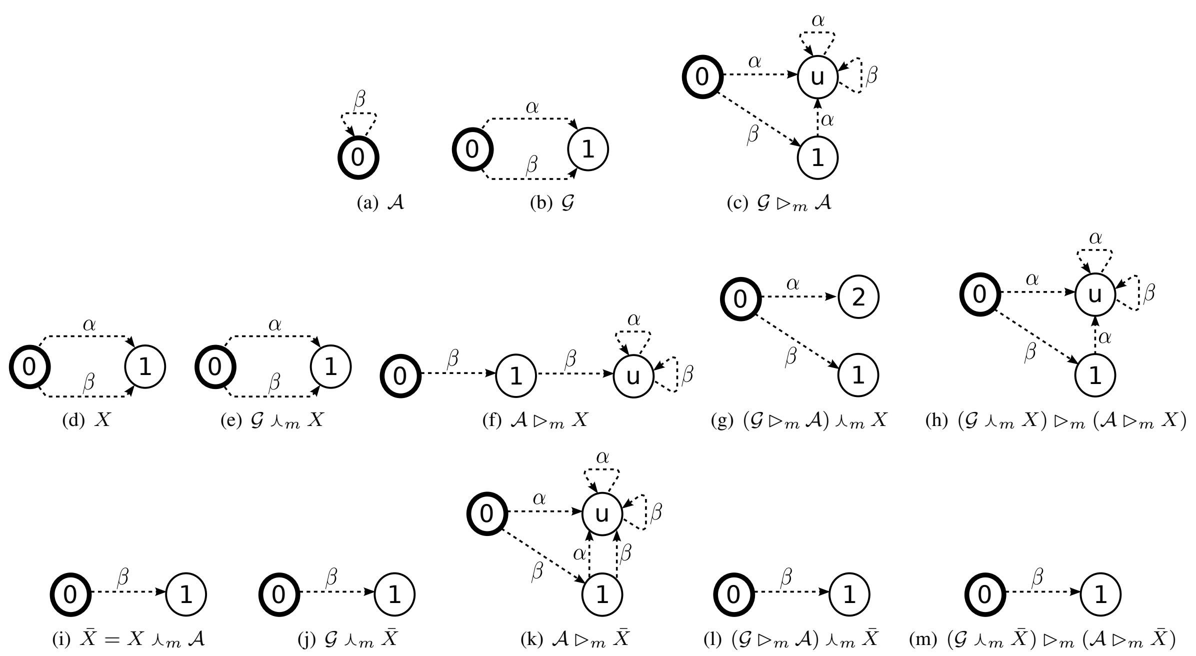
The following theorem affirms our above observation and provides a way to synthesizing modal contracts. Note that as mentioned previously, a pruning procedure is invoked after every combining operation, e.g. it is invoked two times on the left side and three times on the right side of the theorem.

Theorem 3. $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X} = (\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$.

Proof: To prove the satisfaction of Equation (1), we show that every path in $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$ can be simulated by $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$ and vice versa.

- Let p_l be a path in $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$:

  $p_l : ((g_0, a_0), \bar{x_0}) \overset{\alpha_0}{\dashrightarrow} ((g_1, a_1), \bar{x_1}) \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, a_n), \bar{x_n})$.


(a) $\mathcal{A}$ (b) $\mathcal{G}$ (c) $\mathcal{G} \rhd_m \mathcal{A}$

(d) X (e) $\mathcal{G} \curlywedge_m X$ (f) $\mathcal{A} \rhd_m X$ (g) $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m X$ (h) $(\mathcal{G} \curlywedge_m X) \rhd_m (\mathcal{A} \rhd_m X)$

(i) $\bar{X} = X \curlywedge_m \mathcal{A}$ (j) $\mathcal{G} \curlywedge_m \bar{X}$ (k) $\mathcal{A} \rhd_m \bar{X}$ (l) $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$ (m) $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$

Fig. 2: A modal contract over the set of action $\Sigma = \{\alpha, \beta\}$

Then by definition of $\curlywedge_m$, there exist p_{ga} in $(\mathcal{G} \rhd_m \mathcal{A})$, $p_{\bar{x}}$ in $\bar{X}$ and p_a in $\mathcal{A}$:

$$p_{ga} : (g_0, a_0) \overset{\alpha_0}{\dashrightarrow} (g_1, a_1) \ldots \overset{\alpha_n}{\dashrightarrow} (g_n, a_n)$$
$$p_{\bar{x}} : \bar{x}_0 \overset{\alpha_0}{\dashrightarrow} \bar{x}_1 \ldots \overset{\alpha_n}{\dashrightarrow} \bar{x}_n$$
$$p_a : a_0 \overset{\alpha_0}{\dashrightarrow} a_1 \ldots \overset{\alpha_n}{\dashrightarrow} a_n.$$

By definition of $\rhd_m$, the existence of p_{ga} and p_a implies that of path p_g in $\mathcal{G}$:

$$p_g : g_0 \overset{\alpha_0}{\dashrightarrow} g_1 \ldots \overset{\alpha_n}{\dashrightarrow} g_n.$$

Next $p_g, p_{\bar{x}}$ and p_a implies the existence of path p_r in $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$:

$$p_r : ((g_0, \bar{x}_0), (a_0, \bar{x}_0)) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, \bar{x}_n), (a_n, \bar{x}_n)).$$

In addition, assume there is a must transition

$$((g_i, a_i), \bar{x}_i) \xrightarrow{\alpha_i} ((g_{i+1}, a_{i+1}), \bar{x}_{i+1})$$

somewhere in p_l. By definition of $\curlywedge_m$, either

$$(g_i, a_i) \xrightarrow{\alpha_i} (g_{i+1}, a_{i+1}) \text{ holds, or}$$
$$\bar{x}_i \xrightarrow{\alpha_i} \bar{x}_{i+1} \text{ holds,}$$

and implies $(g_i, \bar{x}_i) \xrightarrow{\alpha_i} (g_{i+1}, \bar{x}_{i+1})$. Thus there is also the following must transition in p_r:

$$((g_i, \bar{x}_i), (a_i, \bar{x}_i)) \xrightarrow{\alpha_i} ((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}_{i+1}))$$

- Let p_r be a path in $(\mathcal{G} \curlywedge_m \bar{X}) \rhd_m (\mathcal{A} \rhd_m \bar{X})$:

$$p_r : ((g_0, \bar{x}_0), (a_0, \bar{x}'_0)) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, \bar{x}_n), (a_n, \bar{x}'_n))$$

By induction, we prove that:

$\forall 0 \leq i \leq n$:
$$((g_i, \bar{x}_i), (a_i, \bar{x}'_i)) \text{ is not universal, and } \bar{x}_i \equiv \bar{x}'_i.$$

- Base case $i = 0$: trivial.
- Step case: assume the induction holds up to the i-th state of p_r. By contraposition, assume the $(i+1)$-th state, i.e., $((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}'_{i+1}))$, is universal. Then by definition of $\rhd_m$, the following must hold:

  $$(a_i, \bar{x}_i) \overset{\alpha_i}{\not\dashrightarrow}$$

  which implies $a_i \overset{\alpha_i}{\not\dashrightarrow}$ and $\bar{x}_i \overset{\alpha_i}{\dashrightarrow} \bar{x}_{i+1}$.
  As $\bar{X} = X \curlywedge_m \mathcal{A}$, the latter then implies $a_i \overset{\alpha_i}{\dashrightarrow} a_{i+1}$ by definition of $\curlywedge_m$, contradicting with the former. Thus the $(i+1)$-th state of p_r is not universal and this implies, by definition of $\rhd_m$, that

  $$(a_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (a_{i+1}, \bar{x}'_{i+1})$$
  $$(g_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (g_{i+1}, \bar{x}_{i+1})$$

  which then implies $(a_i, \bar{x}_i) \overset{\alpha_i}{\dashrightarrow} (a_{i+1}, \bar{x}_{i+1})$. Hence, $\bar{x}_{i+1} \equiv \bar{x}'_{i+1}$ by the deterministic assumption on modal automata.

The induction also infers the existence of p_g in $\mathcal{G}$, $p_{\bar{x}}$ in $\bar{X}$, p_a in $\mathcal{A}$:

$$p_g : g_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} g_n$$
$$p_{\bar{x}} : \bar{x}_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} \bar{x}_n$$
$$p_a : a_0 \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} a_n$$

which together implies that of p_l in $(\mathcal{G} \rhd_m \mathcal{A}) \curlywedge_m \bar{X}$:

$$p_l : ((g_0, a_0), \bar{x_0}) \overset{\alpha_0}{\dashrightarrow} \ldots \overset{\alpha_n}{\dashrightarrow} ((g_n, a_n), \bar{x_n}).$$

In addition, if there is a must transition somewhere in p_r:

$$((g_i, \bar{x}_i), (a_i, \bar{x}_i)) \overset{\alpha_i}{\to} ((g_{i+1}, \bar{x}_{i+1}), (a_{i+1}, \bar{x}_{i+1})),$$

then by definition of $\triangleright_m$, there must be

$$(g_i, \bar{x}_i) \overset{\alpha_i}{\to} (g_{i+1}, \bar{x}_{i+1}).$$

Thus either $g_i \overset{\alpha_i}{\to} g_{i+1}$ or $\bar{x}_i \overset{\alpha_i}{\to} \bar{x}_{i+1}$ holds and implies

$$((g_i, a_i), \bar{x}_i) \overset{\alpha_i}{\to} ((g_{i+1}, a_{i+1}), \bar{x}_{i+1})$$

in p_l.

■

With this theorem, our synthesis strategy for modal contracts needs only a minor modification. That is, we compute and use $\bar{X}$ and $\bar{Y}_i$ instead of X and Y_i where

$$\bar{X} \overset{\text{def}}{=} X \curlywedge_m \mathcal{A}_k,$$
$$\bar{Y}_i \overset{\text{def}}{=} Y_i \curlywedge_m \mathcal{A}_{k_i}.$$

in applying equations (2) and (3).

Example 2. We consider the simple message system `System` studied by Bauer et al. [2]. The system consists of two components: component `Server` and component `User`. Their contracts are defined over the action set $\Sigma = \{\texttt{msg}, \texttt{secret\_msg}, \texttt{auth}, \texttt{send}\}$ and are shown in Fig. 3(a)-Fig. 3(h), where *may* transitions underlying *must* transitions are not drawn for simplicity.

The contract $\mathcal{C}_{Server} = (\mathcal{A}_{Server}, \mathcal{G}_{Server})$ models a simple protocol of sending (`send`) a message (`msg`) or secret message (`secret_msg`) from the `Server` to the `User`. In addition, the `Server` waits for an authentication code (`auth`) from the `User` before sending a secret message to it. The authentication code, however, is not required for sending a normal message. The contract $\mathcal{C}_{User} = (\mathcal{A}_{User}, \mathcal{G}_{User})$ then guarantees that the messages can always be received but does not ensure that authentication codes can be sent. The contracts described in this example are almost identical to those provided by Bauer et al. [2] except that of component `Server`, where we make a minor modification to the assumption. That is, authentication codes can only be received before messages are sent to the `User` while they are also allowed after such message sending in the original contract. Therefore, our modified assumptions is stronger than the original one. Decomposing the message system into these two components is then only possible when the system contract $\mathcal{C}_{System} = (\mathcal{A}_{System}, \mathcal{G}_{System})$ can also be decomposed into the component contracts $\mathcal{C}_{Server}$ and $\mathcal{C}_{User}$.

To verify the decomposition, we first normalize all the guarantees as in Fig. 3(c), Fig. 3(e) and Fig. 3(h). We next observe that the composition of the `Server` and `User` normalized guarantees, i.e., $\mathcal{G}^{\text{n}}_{Server} \parallel_m \mathcal{G}^{\text{n}}_{User}$ does not refine $\mathcal{G}^{\text{n}}_{System}$ since the authentication code reception is allowed by the former and not allowed by the latter. In fact, the modification that we made to the contract assumption of component `Server` is the main reason for the failure of this decomposition. Thereby decomposing the message system into the two components would not be possible without performing some corrective synthesis.

We then apply our synthesis strategy in Sect. V to synthesize the `Server` contract w.r.t. $\bar{X}$:

$$\bar{X} = (\mathcal{G}^{\text{n}}_{System} /_m \mathcal{G}^{\text{n}}_{User}) \curlywedge_m \mathcal{A}_{Server}$$

which is shown in Fig. 3(i). The newly-synthesized `Server` contract provides the same guarantee under a more general assumption (Fig. 3(j)). This new assumption corrects our wrong modification and allows authentication codes to always be received. We can now verify easily that the composition of the new `Server` contract and the `User` contract refines the overall `System` contract. As a result, the message system can be obtained by composing components `Server` and `User`.

VIII. Conclusions

In contract-based design, the top-down decomposition of a system into subcomponents is possible when the system general requirement $\mathcal{C}$ can be decomposed into requirements $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ of the subcomponents. To support this top-down design procedure, we have presented a set of decomposing conditions for verifying the decomposition of a contract into a set of contracts. The conditions are defined on top of specifications operators such as normalization, composition and refinement and work for any generic contract framework equipped with such operators. To provide for a complete design methodology, we have also proposed a synthesis strategy which can correct wrong contracts causing the condition failure. Our synthesis strategy can be applied to contract frameworks under the assumption of commutative normalization and conjunction. Although such assumption appears to be a limitation, it is a desirable property for flexible design methodologies as it comes with a synthesis strategy for fixing wrong decompositions. The assumption can be made satisfied by continuously strengthening the core operand.

Our future work includes the implementation of our decomposition and synthesis strategy and the evaluation on verification performance. In particular, one essential step will be the integration of our strategy with several of the contract refinement checking methods that have been proposed in the literature.

References

[1] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "METROII: A design environment for cyber-physical systems," *ACM Trans. on Embedded Computing Systems*, vol. 12, no. 1s, pp. 49:1–49:31, March 2013.

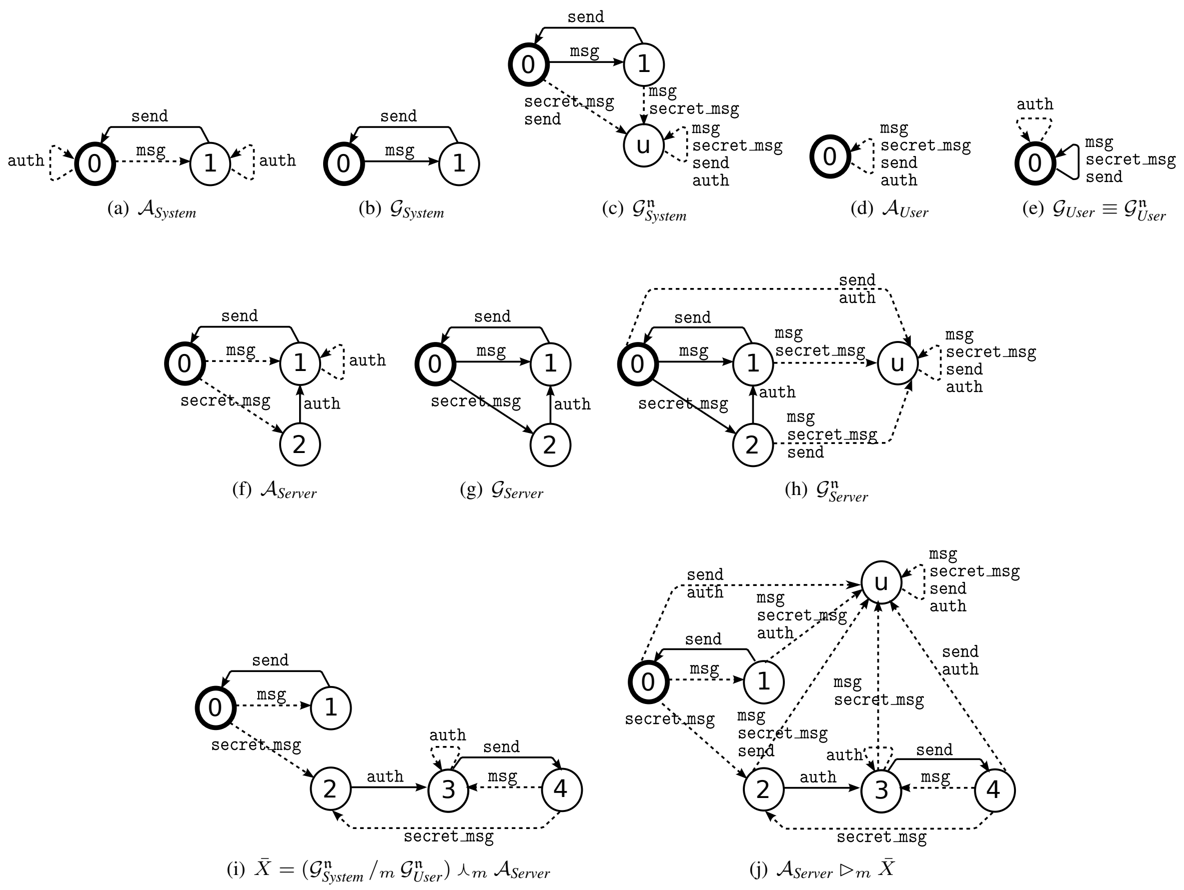
[2] S. Bauer, A. David, R. Hennicker, K. Guldstrand Larsen, A. Legay, U. Nyman, and A. Wsowski, "Moving from specifications to contracts in component-based design," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7212, pp. 43–58.

[3] B. Meyer, "Applying "Design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[4] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. ESEC/FSE-9. New York, NY, USA: ACM, 2001, pp. 109–120.

[5] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.

[6] E. W. Dijkstra, "Guarded commands, non-determinacy and a calculus for the derivation of programs," in *Proceedings of the International Conference on Reliable Software*. ACM, 1975, pp. 2–2.13.



(a) $\mathcal{A}_{System}$ (b) $\mathcal{G}_{System}$ (c) $\mathcal{G}^{n}_{System}$ (d) $\mathcal{A}_{User}$ (e) $\mathcal{G}_{User} \equiv \mathcal{G}^{n}_{User}$

(f) $\mathcal{A}_{Server}$ (g) $\mathcal{G}_{Server}$ (h) $\mathcal{G}^{n}_{Server}$

(i) $\bar{X} = (\mathcal{G}^{n}_{System} /_m \mathcal{G}^{n}_{User}) \curlywedge_m \mathcal{A}_{Server}$ (j) $\mathcal{A}_{Server} \rhd_m \bar{X}$

Fig. 3: Modal contracts for a simple message system

[7] L. Lamport, "Win and sin: Predicate transformers for concurrency," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, pp. 396–428, Jul. 1990.

[8] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *Formal Methods for Components and Objects*, ser. Lecture Notes in Computer Science, F. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Springer Berlin Heidelberg, 2008, vol. 5382, pp. 200–225.

[9] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis, "A contract-based formalism for the specification of heterogeneous systems," in *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, Sept 2008, pp. 142–147.

[10] P. López Martnez and T. Vardanega, "Handling synchronization requirements under separation of concerns in model-driven component-based development," in *Reliable Software Technologies Ada-Europe 2012*, ser. LNCS. Springer Berlin Heidelberg, 2012, vol. 7308, pp. 89–104.

[11] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone, "A modal interface theory for component-based design," *Fundamenta Informaticae*, vol. 108, no. 1–2, pp. 119–149, 2011.

[12] S. Graf, R. Passerone, and S. Quinton, "Contract-based reasoning for component systems with rich interactions," in *Embedded Systems Development: From Functional Models to Implementations*, ser. Embedded Systems. Springer New York, 2014, vol. 20, ch. 8, pp. 139–154.

[13] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed I/O automata: a complete specification theory for real-time systems," in *Proc. of the 13th ACM international conference on Hybrid systems: computation and control*, ser. HSCC '10, Stockholm, Sweden, 2010.

[14] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa, "Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis," in *Reachability Problems*, ser. Lecture Notes in Computer Science, A. Finkel, J. Leroux, and I. Potapov, Eds. Springer Berlin Heidelberg, 2012, vol. 7550, pp. 79–91.

[15] A. Iannopollo, P. Nuzzo, S. Tripakis, and A. Sangiovanni-Vincentelli, "Library-based scalable refinement checking for contract-based design," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.

[16] A. Cimatti and S. Tonetta, "A property-based proof system for contract-based design," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, Sept 2012, pp. 21–28.

[17] W. P. de Roever, "The quest for compositionality," in *Proceedings of IFIP Working Conference on the Role of Abstract Models in Computer Science*. Elsevier Science B.V., 1985.

[18] S.-W. Lin and P.-A. Hsiung, "Counterexample-guided assume-guarantee synthesis through learning," *IEEE Transactions on Computers*, vol. 60, no. 5, pp. 734–750, 2011.

[19] K. Chatterjee and T. Henzinger, "Assume-guarantee synthesis," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4424, pp. 261–275.

[20] M. Bordin and T. Vardanega, "Correctness by construction for high-integrity real-time systems: A metamodel-driven approach," in *Reliable Software Technologies Ada-Europe 2007*, ser. LNCS. Springer Berlin Heidelberg, 2007, vol. 4498, pp. 114–127.

[21] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *Design, Automation and Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.