Chapter 8
Contract-Based Reasoning for Component Systems with Rich Interactions

Susanne Graf, Roberto Passerone and Sophie Quinton

Abstract In this chapter we propose a rule unifying circular and non-circular assume-guarantee reasoning and show its interest for contract-based design and verification. Our work was motivated by the need to combine, in the top-down methodology of the FP7 SPEEDS project, partial tool chains for two component frameworks derived from the HRC model and using different refinement relations. While the L0 framework is based on a simple trace-based representation of behaviors and uses set operations for defining refinement, the more elaborated L1 framework offers the possibility to build systems of components with complex interactions. Our approach in L1 is based on circular reasoning and results in a method for checking contract dominance which does not require the explicit composition of contracts. In order to formally relate results obtained in L0 and L1, we provide a definition of the minimal concepts required by a consistent contract theory and propose abstract definitions which smoothly encompass hierarchical components. Finally, using our relaxed rule for circular reasoning, we show how to use together the L0 and L1 refinement relations and as a result their respective tool chains.

S. Graf (✉)
VERIMAG / CNRS, 2 avenue de Vignate, 38610 Gières, France
e-mail: susanne.graf@imag.fr

R. Passerone
DISI / University of Trento,via Sommarive 5, 38123 Trento, Italy
e-mail: roberto.passerone@unitn.it

S. Quinton
IDA / TU Braunschweig, Hans-Sommer-Straße 66, 38106 Braunschweig, Germany
e-mail: quinton@ida.ing.tu-bs.de

8.1 Introduction

Contract and interface frameworks are emerging as the formalism of choice for system designs that require large and distributed teams, or where the supply chain is complex [1–3]. This style of specification is typically employed for top-down design of systems of components, where the system under design is built by a sequence of decomposition and verification steps. In this chapter we present and study some distinctive features of contract theories for frameworks in which the interaction between components is "rich", i.e., more complex than the usual input/output (I/O) communication. One such component framework is BIP [3] which allows multi-party synchronizations scheduled according to priorities. In addition, we show how to combine results obtained using different contract refinement relations.

Our work has its practical motivation in the component framework HRC [2–5] (standing for Heterogeneous Rich Components) defined in the FP7 IP project SPEEDS [6], which has been reused in the FP7 STREP project COMBEST [7] and the ARTEMIS project CESAR [8]. The HRC model defines component properties in terms of extended transition systems and provides several composition models, ranging from low-level semantic composition to composition frameworks underlying the design tools used by system designers. More precisely, HRC is organized around two abstraction levels called L0 and L1 and describing respectively the *core level* and the *analysis tool level* of HRC [9]. That is, L0 determines the expressive power of the entire model and there exist translations from L1 models to L0. On the other hand, L1 extends the core model with concepts such as coordination mechanisms — the rich interactions mentioned in the title. Analysis tools can then take advantage of these additional concepts to make system descriptions more concise and therefore verification more efficient.

Our objective is to allow combined use of synchronous tools like Simulink [10] for L0 and synchronization-based tools like BIP for L1, which have complementary strengths. For example, Simulink is very convenient for modeling physical dynamic systems or streaming applications. In contrast BIP, which encompasses rich interactions, is well adapted for describing the dynamic behavior of sets of components depending on available resources for memory, energy, communication bandwidth etc. We are interested in this chapter in the relation between the L0 and L1 contract frameworks as we want to use verification results established in L1 for further reasoning within L0. The presence of rich interactions in L1 makes contract composition problematic and leads us to focus instead on *circular reasoning*, which allows a component and its environment to be refined concurrently—each one relying on the abstract description of its context—and entails an interesting rule for proving *dominance*, i.e., refinement between contracts. In order to relate L0 and L1, we define a generic contract framework that uses abstract composition operators and thus encompasses a variety of interaction models, including those for L0 and L1. Finally, we show how to use a relaxed rule for circular reasoning to combine partial tool chains for both frameworks into a complete tool chain for our methodology.

To the best of our knowledge, this is the first time that a rule combining different refinement relations is proposed and used to unify two contract frameworks. While circular reasoning has been extensively studied, e.g., in [11, 12], existing work focuses on finding sufficient conditions for soundness of circular reasoning while we focus on how to use circular reasoning in a contract-based methodology. Non-circular assume-guarantee reasoning is also a topic of intense research focused on finding a decomposition of the system that satisfies the strong condition imposed on at least one of its components [13]. Finally, our contract frameworks are related to interface automata [14]. Since de Alfaro and Henzinger's seminal paper many contract and interface theories have been developed for numerous frameworks (see e.g. [15–20] to name just a few). However these theories focus on composition of contracts while we strive to avoid that and furthermore they do not handle rich interactions. Examples include [20, 21] based on modal I/O automata and [16] defining relational interfaces for capturing functional dependencies between inputs and outputs of an interface. Preliminary versions of our contract framework appeared in [22, 23] but did not address the question of combining results obtained for different refinements.

This chapter is structured as follows: Sect. 8.2 describes our design and verification methodology as well as generic definitions of component and contract framework. It then discusses sufficient reasoning rules for establishing dominance without composing contracts. Section 8.3 presents how the proposed approach is applied to the L0 and L1 frameworks. In particular it shows how their different satisfaction relations may be used together using *relaxed circular reasoning* and discusses practical consequences of this result. Section 8.4 concludes the chapter. The proofs of all theorems presented in this paper are described in [24].

8.2 Design Methodology

Our methodology is based on an abstract notion of component. We characterize a component K by its *interface* defined as a set $\mathscr{P}$ of *ports* which describe what can be observed by its environment. We suppose given a global set of ports $Ports$, which all sets of ports in the following are subsets of. In addition, components are also characterized by their *behavior* . At this level of abstraction, we are not concerned with how behaviors are represented and develop our methodology independently of the particular formalism employed. Interactions (potentially complex) between components are expressed using the concept of *glue* operator [25] . A glue defines how the ports of different components are connected and the kind of synchronization and data exchange that may take place. We denote the composition of two components K_1 and K_2 through a glue gl as $gl\{K_1, K_2\}$. The glue must be defined on the union of the ports $\mathscr{P}_1$ and $\mathscr{P}_2$ of the components.

In order to separate the implementation phase of a component from its integration into the system under design, we use *contracts* [5, 22, 26] . A contract for a component K describes the interface $\mathscr{P}$ of K, the interaction between K and its environment E, the expected behavior of E, called the *assumption* A of the contract, and the expected

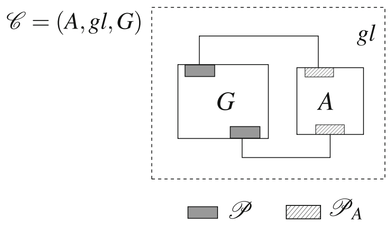


Fig. 8.1 A contract (A, gl, G) for an interface $\mathscr{P}$

behavior of K, called the *guarantee* G. Assumptions and guarantees are in turn expressed as components, defining the interface and the behavior that are considered acceptable from the environment and from the component. Thus, formally, a contract $\mathscr{C}$ for an interface $\mathscr{P}$ is a triple (A, gl, G), where gl is a glue operator on $\mathscr{P} \cup \mathscr{P}_A$ for some $\mathscr{P}_A$ disjoint from $\mathscr{P}$; the assumption A is a component with interface $\mathscr{P}_A$; and the guarantee G is a component with interface $\mathscr{P}$. Note that the interface of the environment is implicitly defined by gl. Graphically, we represent contracts as in Fig. 8.1.

From a macroscopic point of view, we adopt a top-down design and verification methodology (see Fig. 8.2) in which global requirements are pushed progressively from the top-level system to the low-level atomic components. As usual, this is just a convenient representation; in real life, the final picture is always obtained in several iterations alternatively going up and down the hierarchy [27].

While the refinement relation between a specification and an implementation is at the core of component-based design, in contract-based design refinement takes different forms depending on whether it relates a system to a specification, two contracts or an implementation to a contract. In this chapter we use a methodology which divides the design and verification process into three steps corresponding to these three forms of refinement.

We assume that the system K under construction has to realize a global requirement φ together with an environment on which we may have some knowledge, expressed by a property A. Both φ and A are expressed w.r.t. the interface $\mathscr{P}$ of K. We proceed as follows: (1) define a *contract* $\mathscr{C} = (A, gl, G)$ for $\mathscr{P}$ such that $gl\{A, G\}$ *conforms* to φ; (2) decompose K as subcomponents K_i connected through a glue operator gl_I and provide a contract $\mathscr{C}_i$ for each of them; possibly iterate this step if needed; (3) prove that whenever a set of implementations K_i *satisfy* their contracts $\mathscr{C}_i$, then their composition satisfies the top-level contract $\mathscr{C}$ (*dominance*) — and thus guarantee φ; (4) provide such implementations.

The correctness proof for a particular system is therefore split into 3 phases: *conformance* (denoted $\preccurlyeq$) of the system defined by the top-level contract $\mathscr{C}$ to φ; *dominance* of $\mathscr{C}$ by the composition of the set of contracts $\{\mathscr{C}_i\}$ through gl_I; and *satisfaction* (denoted $\models$) of each $\mathscr{C}_i$ by the corresponding implementation K_i. Thus, *conformance* relates closed systems, *dominance* relates contracts, while *satisfaction* relates components to contracts.

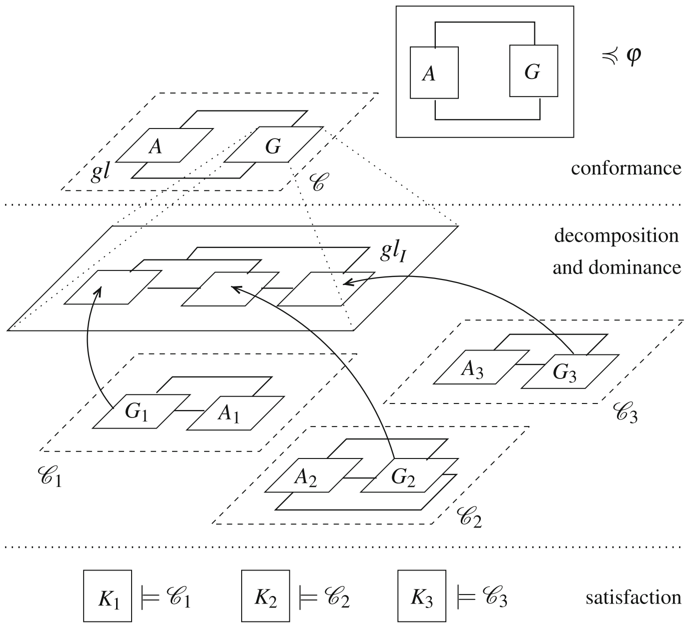


Fig. 8.2 Proof of $gl\{A, gl_I\{K_1, K_2, K_3\}\} \preccurlyeq \varphi$

Note that the assumption of $\mathscr{C}_1$ is represented as one component A_1 while in the actual system K_1 will be used in the context of three components, namely K_2, K_3 and A. Thus, we need to relate the actual glues gl and gl_I to the glue gl_1 of $\mathscr{C}_1$. In other words, we need a glue gl_{E_1} to compose K_2, K_3 and A as well as an operation $\circ$ on glues such that $gl \circ gl_I = gl_1 \circ gl_{E_1}$. In most cases, $\circ$ cannot simply be composition of functions and has to involve some *flattening* of the system.

8.2.1 Contract Framework

To summarize, we consider a component framework that smoothly supports complex composition operators and hierarchical components. The elements of the component framework are as follows:

Definition 1 *(Component framework).* A *component framework* is defined by a tuple $(\mathscr{K}, GL, \circ, \cong)$ where:

- $\mathscr{K}$ is a set of *components*. Each component $K \in \mathscr{K}$ has as *interface* a set of *ports*, denoted $\mathscr{P}_K$ and subset of our global set of ports $Ports$.

- GL is a set of *glues*. A glue is a partial function $2^{\mathscr{K}} \longrightarrow \mathscr{K}$ transforming a set of components into a new composite component. Each $gl \in GL$ is defined on a set of ports S_{gl}, called *support set*, and defines a new interface $\mathscr{P}_{gl}$ for the new component, called *exported interface*. $K = gl(\{K_1, \dots, K_n\})$ is defined if $K_1, \dots, K_n \in \mathscr{K}$ have disjoint interfaces, $S_{gl} = \bigcup_{i=1}^{n} \mathscr{P}_{K_i}$ and $\mathscr{P}_K = \mathscr{P}_{gl}$.
- $\circ$ is a partial operator on GL, called *flattening*, to compose glues. $gl \circ gl'$ is defined if $\mathscr{P}_{gl'} \subseteq S_{gl}$. Its support set is $S_{gl} \setminus \mathscr{P}_{gl'} \cup S_{gl'}$ and its interface is $\mathscr{P}_{gl}$.
- $\cong \ \subseteq \mathscr{K} \times \mathscr{K}$ is an equivalence relation between components.

We simplify our notation by writing $gl\{K_1, \dots, K_n\}$ instead of $gl(\{K_1, \dots, K_n\})$. The equivalence relation $\cong$ is typically used for relating composite components with their semantics given as an atomic component. More importantly, $\circ$ must be coherent with $\cong$ in the sense that $gl\{gl'\{\mathscr{K}_1\}, \mathscr{K}_2\} \cong (gl \circ gl')\{\mathscr{K}_1 \cup \mathscr{K}_2\}$ for any sets of components $\mathscr{K}_i$ such that all terms are defined.

After formalizing generic properties required from a component framework, we now define the relations used in the methodology for dealing with contracts. Satisfaction is usually considered as a derived relation and chosen as the weakest relation implying conformance and preserved by composition. We loosen the coupling between satisfaction and conformance to obtain later stronger reasoning schemata for dominance. Furthermore, we propose a representation of satisfaction as a set of *refinement under context* relations denoted $\sqsubseteq_{A,gl}$ and such that $K \sqsubseteq_{A,gl} G$ iff $K \models (A, gl, G)$.

Definition 2 *(Contract framework)* A *contract framework* is defined by a tuple $(\mathscr{K}, GL, \circ, \cong, \preccurlyeq, \models)$ where:

- $(\mathscr{K}, GL, \circ, \cong)$ is a component framework.
- $\preccurlyeq \subseteq \mathscr{K} \times \mathscr{K}$ is a preorder called *conformance* relating components having the same interface.
- $\models$ is a relation called *satisfaction* between components and contracts s.t.: the relations $\sqsubseteq_{A,gl}$ defined by $K \sqsubseteq_{A,gl} G$ iff $K \models (A, gl, G)$ are preorders; and, if $K \models (A, gl, G)$ then $gl\{A, K\} \preccurlyeq gl\{A, G\}$.

Our definition of satisfaction emphasizes the fact that $\models$ can be seen as a set of refinement relations where $K \sqsubseteq_{A,gl} G$ means that K refines G in the context of A and gl. The condition which relates satisfaction and conformance ensures that the actual system $gl\{A, K\}$ will conform to the global requirement φ discussed in the methodology because $\preccurlyeq$ is transitive and $gl\{A, G\} \preccurlyeq \varphi$.

Example 1 Typical notions of conformance for labeled transition systems are *trace inclusion* and its structural counterpart *simulation*. For these, satisfaction is usually defined as the weakest relation implying conformance.

$$K \models (A, gl, G) \triangleq gl\{K, A\} \preccurlyeq gl\{G, A\}$$

Dominance is a key notion for reasoning about contracts rather than using refinement between components. Proving that a contract $\mathscr{C}$ dominates $\mathscr{C}'$ means showing that every component satisfying $\mathscr{C}$ also satisfies $\mathscr{C}'$.[1] However, a dominance check involves in general not just a pair of contracts: a typical situation would be the one depicted in Fig. 8.2, where a set of contracts $\{\mathscr{C}_i\}_{i=1}^n$ are attached to disjoint interfaces $\{\mathscr{P}_i\}_{i=1}^n$. Besides, a glue gl_I is defined on $P = \bigcup_{i=1}^n \mathscr{P}_i$ and a contract $\mathscr{C}$ is given for P. In this context, a set of contracts $\{\mathscr{C}_i\}_{i=1}^n$ dominates a contract $\mathscr{C}$ w.r.t. a glue gl_I if any set of components satisfying contracts $\mathscr{C}_i$, when composed using gl_I, makes a component satisfying $\mathscr{C}$.

Definition 3 *(Dominance)* Let $\mathscr{C}$ be a contract on $\mathscr{P}$, $\{\mathscr{C}_i\}_{i=1}^n$ a set of contracts on $\mathscr{P}_i$ and gl_I a glue such that $S_{gl_I} = \bigcup_{i=1}^n \mathscr{P}_i$ and $\mathscr{P} = \mathscr{P}_{gl_I}$. Then $\{\mathscr{C}_i\}_{i=1}^n$ *dominates* $\mathscr{C}$ *with respect to* gl_I iff for all components $\{K_i\}_{i=1}^n$:

$$(\forall i : K_i \models \mathscr{C}_i) \implies gl_I\{K_1, \ldots, K_n\} \models \mathscr{C}$$

Note that this formal definition of dominance does not help establishing dominance in practice because looking at all possible components satisfying a contract is not realistic. What we need is a sufficient condition that refers to assumptions and guarantees, rather than components. One such condition exists when the composition of the low-level guarantees G_i satisfies the top-level contract $\mathscr{C}$ and furthermore each low-level assumption A_i is discharged by the abstraction of its environment defined by the guarantees of the other components. Formally:

$$\begin{cases} gl_I\{G_1, \ldots, G_n\} \models \mathscr{C} \\ \forall i : gl_{E_i}\{A, G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n\} \models \mathscr{C}_i^{-1} \end{cases} \tag{8.1}$$

where for any contract $\mathscr{C}_i = (A_i, gl_i, G_i)$ we use the notation $\mathscr{C}_i^{-1}$ to denote the contract (G_i, gl_i, A_i).

In the next subsection, we provide two rules which indeed make the previous condition sufficient for establishing dominance: one is similar to circular assume-guarantee reasoning and the other one deals with preservation of satisfaction by composition. This result is particularly significant because one can check dominance while avoiding composition of contracts, which is impossible in the general case and leads to state explosion in most concrete contract frameworks.

[1] One may also need to ensure that the assumptions of the low-level contracts are indeed satisfied in the actual system. This is achieved by strengthening the definition with:

$$\forall E \text{ on } \mathscr{P}_A, \text{if } E \models (G', gl', A') \text{ then } E \models (G, gl, A)$$

8.2.2 Reasoning Within a Contract Framework

We use here the representation of satisfaction as a set of refinement under context relations $\sqsubseteq_{A,gl}$ where $K \sqsubseteq_{A,gl} G$ if and only if $K \models (A, gl, G)$. The usual non-circular assume-guarantee rule reads as follows in our context:

$$K \sqsubseteq_{A,gl} G \wedge E \sqsubseteq A \implies K \sqsubseteq_{E,gl} G \tag{8.2}$$

where $E \sqsubseteq A$ denotes that for any component G and gl such that $\sqsubseteq_{G,gl}$ is defined $E \sqsubseteq_{G,gl} A$. This rule relates the behavior of K, when composed with the abstract environment A, to the behavior of K, when composed with its actual environment E. However it is quite limited as it imposes a very strong condition on E. Hence the following rule which is commonly referred to as *circular reasoning*.

$$K \sqsubseteq_{A,gl} G \wedge E \sqsubseteq_{G,gl} A \implies K \sqsubseteq_{E,gl} G$$

Note that E and K may symmetrically rely on each other. For a given contract framework, this property can be proven by an induction based on the semantics of composition and refinement. Unfortunately, circular reasoning is not sound in general. In particular it does not hold for parallel composition with synchronizations (as in Petri nets or process algebras) or instantaneous mutual dependencies between inputs and outputs (as in synchronous formalisms). The following example illustrates one possible reason for the non validity of circular reasoning.[2]

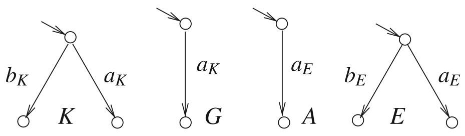
Example 2 Consider a contract framework where components are labeled transition systems and composition is strong synchronization between corresponding labels and interleaving of others denoted $\|$. Define conformance as simulation and satisfaction as the usual relation defined in Example 1. The circular reasoning rule translates into: if $K \| A$ is simulated by $G \| A$ and $E \| G$ is simulated by $A \| G$ then $K \| E$ is simulated by $G \| E$. In the example of Fig. 8.3, both G and A forbid a synchronization between b_K and b_E from occurring. This allows their respective refinements according to $\sqsubseteq^{\preccurlyeq}$, namely K and E, to offer respectively b_K and b_E, since they can rely on respectively G and A to forbid their actual occurrence. But obviously, the composition $K \| E$ now allows a synchronization between b_K and b_E.

Note that this satisfaction relation can be strengthened to obtain a more restrictive relation for which circular reasoning is sound. This is the approach taken for the L1 contract framework in Sect. 8.3.2, where we need circular reasoning to avoid composition of contracts.

A second rule which is used for compositional reasoning in most frameworks is: if $I \sqsubseteq S$, then $I \| E \sqsubseteq S \| E$. It states that if an implementation I refines its specification S then it refines it in any environment E. The equivalent of this rule for satisfaction is more complex as refinement here relates closed systems.

[2] Note that non-determinism is another reason here for the non validity of circular reasoning.

Fig. 8.3 $K \parallel A \preccurlyeq G \parallel A$ and $E \parallel G \preccurlyeq A \parallel G$ but $K \parallel E \not\preccurlyeq G \parallel E$



Definition 4 Satisfaction $\models$ is preserved by composition iff for any component E, gl such that $S_{gl} = \mathscr{P}_E \cup \mathscr{P}$ for some $\mathscr{P}$ such that $\mathscr{P} \cap \mathscr{P}_E = \emptyset$ and gl_E, E_1, E_2 such that $E = gl_E\{E_1, E_2\}$, the following holds for any components I, S on $\mathscr{P}$:

$$I \sqsubseteq_{E,gl} S \implies gl_1\{I, E_1\} \sqsubseteq_{E_2, gl_2} gl_1\{S, E_1\}$$

where gl_1 and gl_2 are such that $gl \circ gl_E = gl_2 \circ gl_1$.

We now have the ingredients to formalize our sufficient condition for dominance. This condition reduces a dominance proof to a set of satisfaction checks, one for the refinement between the guarantees and n for discharging individual assumptions.

Theorem 1 *Suppose that circular reasoning is sound and satisfaction is preserved by composition. If* $\forall i\ \exists gl_{E_i} : gl \circ gl_I = gl_i \circ gl_{E_i}$ *then to prove that* $\{\mathscr{C}_i\}_{i=1}^n$ *dominates* $\mathscr{C}$ *w.r.t. gl, it is sufficient to prove that condition* (1) *holds.*

8.3 Circular Reasoning in Practice

In this section, we show how the results presented in the previous section have been applied within the SPEEDS project: we define two contract frameworks, called L0 and L1, and show how to combine them.

8.3.1 The L0 Framework

A component K with interface $\mathscr{P}_K$ at level L0 of HRC is defined as a set of behaviors in the form of traces, or runs, over $\mathscr{P}_K$. The behaviors correspond to the history of values seen at the ports of the component for each particular behavior. For instance, these histories could be the traces generated by a labeled transition system (LTS). Composition is defined as a composite that retains only the matching behaviors of the components. If the ports of the two components have the same names, composition at the level of trace sets boils down to a simple intersection of the sets of behaviors. Because in our framework components must have disjoint sets of ports under composition, we must introduce glues, or connectors, as explicit components that establish a synchronous relation between the histories of connected ports. The collection of these simple connectors forms the glues $gl \in GL$ of our framework at the L0 level.

We can model a glue as an extra component K_{gl}, whose set of ports includes all the ports of the components involved in the composition. This component has as the set of behaviors all the identity traces. Composition can then be taken as the intersection of the sets of behaviors of the components, together with the glue. To make this work, we must also equalize the ports of all trace sets using inverse projection $\mathbf{proj}^{-1}_{\mathscr{P}_i,\mathscr{P}}$, which extends behaviors over $\mathscr{P}_1$ with the appropriate additional ports of $\mathscr{P}$. If we denote the interface of the composite as $\mathscr{P}_{gl}$, and if $\mathscr{K} = \{K_1, \ldots, K_n\}$ is a set of components such that $\mathscr{P}_1, \ldots, \mathscr{P}_n$ are pairwise disjoint, then a glue gl for $\mathscr{K}$ is a component K_{gl} defined on the ports $\mathscr{P} = \mathscr{P}_{gl} \cup (\bigcup_{i=1}^{n} \mathscr{P}_i)$, and:

$$\begin{aligned} K &= gl\{K_1, \ldots, K_n\} \\ &= \mathbf{proj}_{p_{gl},\mathscr{P}} \left(K_{gl} \cap \mathbf{proj}^{-1}_{\mathscr{P}_1,\mathscr{P}} (K_1) \cap \cdots \cap \mathbf{proj}^{-1}_{\mathscr{P}_n,\mathscr{P}} (K_n) \right) \end{aligned}$$

The definition of $\circ$ is straightforward: since glues are themselves components, their composition follows the same principle as component composition. Finally, the $\cong$ relation on $\mathscr{K}$ is taken as equality of sets of traces.

In the L0 model there exists a unique maximal component satisfying a contract $\mathscr{C}$, namely $M_{\mathscr{C}} = G \cup \neg A$, where $\neg$ denotes the operation of complementation on the set of all behaviors over ports $\mathscr{P}_A$. A contract $\mathscr{C} = (A, G)$ is in *canonical form* when $G = M_{\mathscr{C}}$. Every contract has an equivalent contract in canonical form, which is obtained by replacing G with $M_{\mathscr{C}}$. The operation of computing a canonical form is well defined, since the maximal implementation is unique, and it is idempotent. It is easy to show that $K \models \mathscr{C}$ if and only if $K \subseteq M_{\mathscr{C}}$.

The L0 contract framework has strong compositional properties, which derive from its simple definition and operators [26]. The theory, however, depends on the effectiveness of certain operators, complementation in particular, which are necessary for the computation of canonical forms. While the complete theory can be formulated without the use of canonical forms, complementation remains fundamental in the definition of contract composition, which is at the basis of system construction. Circular reasoning is not sound for contracts which are *not* in canonical form (Example 2 is a counter-example in that case). This is a limitation of the L0 framework, since working with canonical forms could prove computationally hard.

8.3.2 The L1 Framework

L1 composition is based on *interactions*, which involve non-empty sets of ports. An interaction is defined by the components that synchronize when it takes place and the ports through which these components synchronize. Interactions are structured into connectors which are used as a mechanism for *encapsulation*: only these connectors appear at the interface of a composite component. This enables to abstract the behavior of a component in a black-box manner, by describing which connector is triggered but not exactly which interaction takes place. Furthermore L1 is expressive enough to encompass synchronous systems.

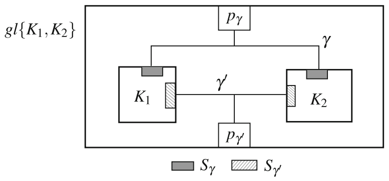


Fig. 8.4 The role of connectors in a composition

Definition 5 An *atomic component* on an interface $\mathscr{P}$ is defined by an LTS $K = (Q, q^0, 2^{\mathscr{P}}, \longrightarrow)$, where Q is a set of states, q^0 an initial state and $\longrightarrow \ \subseteq Q \times 2^{\mathscr{P}} \times Q$ is a transition relation.

Note that atomic components are labeled by sets of ports rather than ports because we allow several ports of a component to be triggered at the same time.

Definition 6 An *interaction* is a non-empty set of ports. A *connector* γ is defined by a set of ports S_γ called the *support set* of γ, a port p_γ called its *exported port* and a set $\mathcal{I}(\gamma)$ of interactions in S_γ.

The notions of support set and exported port are illustrated in Fig. 8.4, where connectors relate in a composition a set of inner ports (of the subcomponents) to an outer port (of the composite component). One should keep in mind that a connector γ, and thus the exported port p_γ, represents a set of interactions rather than a single interaction.

Typical connectors represent rendezvous (only one interaction, equal to the support set), broadcast (all the interactions containing a specific port called *trigger*) and also mutual exclusion (some interactions but not their union).

We now define glues as sets of connectors which may be used together in order to compose components.

Definition 7 A glue gl on a support set S_{gl} is a set of connectors with distinct exported ports and with support sets included in S_{gl}.

A glue gl defines as exported interface $\mathscr{P}_{gl}$ the set $\{p_\gamma \mid \gamma \in gl\}$. Besides, $\mathcal{I}(gl)$ denotes the set of all interactions of the connectors in gl, i.e.: $\mathcal{I}(gl) = \bigcup_{\gamma \in gl} \mathcal{I}(\gamma)$. In Fig. 8.4, gl is composed of connectors γ and γ' and defines a composite component denoted $gl\{K_1, K_2\}$.

Definition 8 A *component* is either an atomic component or it is inductively defined as the composition of a set of components $\{K_i\}_{i=1}^n$ with disjoint interfaces $\{\mathscr{P}_i\}_{i=1}^n$ using a glue gl on $\mathscr{P} = \bigcup_{i=1}^n \mathscr{P}_i$. Such a composition is called a *composite component* on $\mathscr{P}_{gl}$ and it is denoted $gl\{K_i\}_{i=1}^n$.

So far, we have defined components and glues. Glues can be composed so as to allow flattening of components. Such a composition requires to handle *hierarchical connectors* built by merging connectors defined at different levels of hierarchy. The definition of the operator $\circ$ used for this purpose is omitted here and can be found in [24]. Connectors whose exported ports and support sets are not related are called *disjoint* and need not be composed. The operator $\circ$ is then easily extended to glues:

the composition $gl \circ gl'$ of two glues gl and gl' is obtained from $gl \cup gl'$ by inductively composing all connectors which are not disjoint.

We can now formally define the flattened form of a component. This in turn will allow us to provide an equivalence relation between components based on the semantics of their flattened form. A component is called *flat* if it is atomic or of the form $gl\{K_1, \ldots, K_n\}$, where all K_i are atomic components. A component that is not flat is called *hierarchical*. A hierarchical component K is of the form $gl\{K_1, \ldots, K_n\}$ such that at least one K_i is composite. Thus, such a K can be represented as $gl\{gl'\{\mathscr{K}^1\}, \mathscr{K}^2\}$, where $\mathscr{K}^1$ and $\mathscr{K}^2$ are sets of components.

Definition 9 The *flattened form* of a component K is denoted $\flat(K)$ and defined inductively as:

- if K is a flat component, then $\flat(K)$ is equal to K.
- otherwise, K is of the form $gl\{gl'\{\mathscr{K}^1\}, \mathscr{K}^2\}$, and then $\flat(K)$ is the flattened form of $(gl \circ gl')\{\mathscr{K}^1 \cup \mathscr{K}^2\}$.

Definition 10 The *semantics* $[\![K]\!]$ of a flat component $K = gl\{K_1, \ldots, K_n\}$ is defined as $(Q, q^0, \mathfrak{I}(gl), \longrightarrow)$, where $Q = \prod_{i=1}^n Q_i$, $q^0 = (q_1^0, \ldots, q_n^0)$ and $\longrightarrow$ is such that: given two states $q^1 = (q_1^1, \ldots, q_n^1)$ and $q^2 = (q_1^2, \ldots, q_n^2)$ in Q and an interaction $\alpha \in \mathfrak{I}(gl)$, $q^1 \xrightarrow{\alpha} q^2$ if and only if $\forall i, q_i^1 \xrightarrow{\alpha_i}_i q_i^2$, where $\alpha_i = \alpha \cap \mathscr{P}_i$.

We use the convention that $\forall q : q \xrightarrow{\emptyset} q$ so components not involved in an interaction do not move. Thus the semantics of a flat component is obtained as the composition of its constituting LTS where labels are synchronized according to the interactions of $\mathfrak{I}(gl)$.

We then define equivalence $\cong$ as follows: two components are equivalent if their flattened forms have the same semantics. Note that in practice one would prefer to define the semantics of a hierarchical component as a function of the semantics of its constituting components. In presence of encapsulation this requires to distinguish between closed and open systems and thus to provide two different semantics. Details can be found in [24].

We now have the ingredients for defining the L1 component framework and we focus on its contract framework.

Definition 11 $K_1 \preccurlyeq^{L1} K_2$ if and only if $[\![K_1]\!]$ is simulated by $[\![K_2]\!]$.

Thus L1-conformance is identical to L0-conformance for components without non-observable non-determinism, and otherwise stronger. Note that in verification tools, in order to check trace inclusion efficiently, one will generally check simulation anyway. Satisfaction is defined as follows.

Definition 12 A component K *satisfies* a contract $\mathscr{C} = (A, gl, G)$ for $\mathscr{P}_K$, denoted $K \models^{L1} (A, gl, G)$, if and only if:

$$\begin{cases} gl\{K, A_{det}\} \preccurlyeq^{L1} gl\{G, A_{det}\} \\ (q_K, q_A)\, \mathfrak{R}\, (q_G, q'_A) \wedge \exists q'_K : q_K \xrightarrow{\alpha}_K q'_K \implies \exists q'_G : q_G \xrightarrow{\alpha}_G q'_G \end{cases}$$

where A_{det} is the determinization of A, $\mathfrak{R}$ is the relation on states proving that $gl\{K, A_{det}\} \preccurlyeq^{L1} gl\{G, A_{det}\}$ and $\alpha \in 2^{\mathscr{P}_K}$ is such that $\exists \alpha' \in \mathfrak{I}(gl) : \alpha \subseteq \alpha'$.

Thus $\models^{L1}$ strengthens the satisfaction relation used in the L0 framework by: 1) determinizing A; 2) requiring every transition of K to have a counterpart in each related state of G — unless it is structurally forbidden by gl — but the target states of the transition need to be related only if the environment allows this transition. As a consequence, $\models^{L1}$ allows circular reasoning.

8.3.3 Relaxed Circular Reasoning

We have presented in the previous sections two contract frameworks developed in the SPEEDS project. We show now how we use their respective tool chains together. Unifying the L0 and L1 component frameworks is quite straightforward. Nevertheless, we have introduced two different notions of satisfaction: $\models^{L0}$ and $\models^{L1}$ where the second one is strictly stronger than the first one. To combine results based on L0 and L1, we propose a rule called *relaxed circular reasoning* for two (possibly different) refinement relations:

$$K \sqsubseteq^1_{A,gl} G \wedge E \sqsubseteq^2_{G,gl} A \implies K \sqsubseteq^1_{E,gl} G \tag{8.3}$$

This rule generalizes circular and non-circular reasoning by not restricting $\sqsubseteq^2_{G,gl}$ to refinement under context $\sqsubseteq^1_{G,gl}$ or refinement in any context $\sqsubseteq^1$. Depending on which relation is the most restrictive it can be used in two different ways:

1. If the first relation allows circular reasoning and is stronger than the second one (i.e. $K \sqsubseteq^1_{A,gl} G \implies K \sqsubseteq^2_{A,gl} G$) then our new rule relaxes circular reasoning by requiring $E \sqsubseteq^2_{G,gl} A$ rather than $E \sqsubseteq^1_{G,gl} A$.
2. Symmetrically, if the first relation does not allow circular reasoning and refinement in any context $\sqsubseteq^1$ is stronger than the second one then this rule relaxes non circular reasoning by requiring $E \sqsubseteq^2_{G,gl} A$ rather than $E \sqsubseteq^1 A$.

Interestingly, relaxed circular reasoning can be used both ways for L0- and L1-satisfaction. First it leads to a relaxed sufficient condition for dominance in L1.

Theorem 2 *$K \sqsubseteq^{L1}_{A,gl} G \wedge E \sqsubseteq^{L0}_{G,gl} A$ implies $K \sqsubseteq^{L1}_{E,gl} G$.*

Theorem 3 *If $\forall i\ \exists gl_{E_i} : gl \circ gl_I = gl_i \circ gl_{E_i}$ the following is sufficient to prove that $\mathscr{C}$ dominates $\{\mathscr{C}_i\}_{i=1..n}$ w.r.t. gl:*

$$\begin{cases} gl_I\{G_1, \ldots, G_n\} \models^{L1} \mathscr{C} \\ \forall i : gl_{E_i}\{A, G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n\} \models^{L0} \mathscr{C}_i^{-1} \end{cases}$$

In that case, checking that contracts $\{\mathscr{C}_i\}_{i=1}^n$ L1-dominate a contract $\mathscr{C}$ requires one L1-satisfaction check and n L0-satisfaction checks. This is particularly interesting since checking L0-satisfaction may be achieved by using other tools or approaches (that may not need circular reasoning). Moreover, dominance can be established more often as L1-satisfaction is stronger than L0-satisfaction. Second:

Theorem 4 $K \sqsubseteq^{L0}_{A,gl} G \wedge E \sqsubseteq^{L1}_{G,gl} A$ *implies* $K \sqsubseteq^{L0}_{E,gl} G$.

This result made it possible in SPEEDS to incorporate results from tools checking L0-satisfaction with results obtained through L1-dominance (implemented by a set of L1-satisfaction checks), thus building a complete tool chain.

8.4 Conclusion and Future Work

The work presented in this chapter has been motivated by the necessity of combining contract-based verification tools and corresponding results for two component frameworks L0 and L1 defined in the context of the European SPEEDS project. In particular, we were interested in using dominance results established in L1 – and which cannot be obtained using the L0 refinement relation – for further reasoning in L0. To that purpose, we have presented an abstract notion of contract framework for a given component framework that defines three different notions of refinement, that is, conformance, dominance and satisfaction. We show how to derive these notions from refinement of closed systems and refinement under context and we provide a methodology for compositional and hierarchical verification of global properties.

We have studied circular reasoning as a powerful means for proving dominance. As circular reasoning does not always hold for usual notions of refinement, we provide proof rules for dominance relying on a relaxed notion of circular reasoning based on two notions of refinement. We have then shown that our abstract framework is general enough to represent both L0 and L1 as specific instances and proved that the L0 and L1 refinement relations satisfy the condition for relaxed circular reasoning.

This approach was applied to only simple case studies in the SPEEDS project and should therefore rather be seen as a proof of concept. The practical relevance of such an approach is that it opens up ways of connecting tools that work at different levels of abstraction, and relate their results to prove stronger properties. In addition, our results relax the requirements on the tools, since circular reasoning would not be needed at the L0 level.

Acknowledgments This work was supported in part by the EU projects COMBEST (n. 215543) and ArtistDesign (n. 214373).

References

1. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. J. Control **18**(3), 217–238 (2012). doi:10.3166/EJC.18.217-238

2. Damm, W.: Controlling speculative design processes using rich component models. In: Proceedings of ACSD'05, pp. 118–119. IEEE Computer Society (2005)
3. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Proceedings of SEFM'06, pp. 3–12. IEEE Computer Society (2006)
4. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: F.S. de Boer, M.M. Bonsangue, S. Graf, Willem-Paul de Roever (eds.) Formal Methods for Components and Objects, 6th International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007, Revised Papers, Lecture Notes in Computer Science, vol. 5382, pp. 200–225. Springer (2008). doi: 10.1007/978-3-540-92188-2
5. Benvenuti, L., Ferrari, A., Mangeruca, L., Mazzi, E., Passerone, R., Sofronis, C.: A contract-based formalism for the specification of heterogeneous systems. In: Proceedings of the Forum on Specification, Verification and Design Languages (FDL08), pp. 142–147. Stuttgart, Germany (2008). doi: 10.1109/FDL.2008.4641436
6. SPEEDS Consortium: Home page. http://www.speeds.eu.com
7. COMBEST Consortium: Home page. http://www.combest.eu
8. CESAR Consortium: Home page. http://www.cesarproject.eu/
9. Partners, S.: SPEEDS metamodel. SPEEDS project deliverable D2.1.5 (2009)
10. The Mathworks, Inc.: MATLAB simulink. http://www.mathworks.com
11. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods Syst. Des. **15**(1), 7–48 (1999)
12. Maier, P.: A lattice-theoretic framework for circular assume-guarantee reasoning. Ph.D. thesis, Universität des Saarlandes (2003)
13. Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Trans. Softw. Eng. Methodol. **1**7(2) (2008)
14. Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of ESEC/SIGSOFT FSE'01, pp. 109–120. ACM Press (2001)
15. Larsen, K.G., Nyman, U., Wasowski, A.: Interface input/output automata. In: Proceedings of FM'06, LNCS, vol. 4085, pp. 82–97 (2006)
16. Tripakis, S., Lickly, B., Henzinger, T.A., Lee, E.A.: On relational interfaces. In: Proceedings of EMSOFT'09, pp. 67–76 (2009)
17. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In: Proceedings of ACSD'10, pp. 223–232 (2010)
18. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: Modal interfaces: Unifying interface automata and modal specifications. In: Proceedings of the Ninth International Conference on Embedded Software (EMSOFT09), pp. 87–96. Grenoble, France (2009)
19. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Passerone, R.: Why are modalities good for Interface Theories? In: Proceedings of the Ninth International Conference on Application of Concurrency to System Design (ACSD09), pp. 119–127. Augsburg, Germany (2009)
20. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: A modal interface theory for component-based design. Fundamenta Informaticae 108(1–2), 119–149 (2011). 10.3233/FI-2011-416
21. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O automata for interface and product line theories. In: Proceedings of ESOP'07, LNCS, vol. 4421, pp. 64–79 (2007)
22. Quinton, S., Graf, S.: Contract-based verification of hierarchical systems of components. In: Proceedings of SEFM'08, pp. 377–381. IEEE Computer Society (2008)
23. Hafaiedh, I.B., Graf, S., Quinton, S.: Reasoning about safety and progress using contracts. In: Proceedings of ICFEM'10, pp. 436–451 (2010)
24. Graf, S., Passerone, R., Quinton, S.: Contract-based reasoning for component systems with complex interactions. Research report TR-2010-12, VERIMAG (2010 updated 2013)
25. Sifakis, J.: A framework for component-based construction. In: Proceedings of SEFM'05, pp. 293–300. IEEE Computer Society (2005)

26. Benveniste, A., Caillaud, B., Passerone, R.: A generic model of contracts for embedded systems. Rapport de recherche 6214, Institut National de Recherche en Informatique et en Automatique (2007)
27. Pinto, A., Bonivento, A., Sangiovanni-Vincentelli, A.L., Passerone, R., Sgroi, M.: System level design paradigms: Platform-based design and communication synthesis. ACM Trans. Des. Autom. Electron. Syst. **1**1(3), 537–563 (2006). http://doi.acm.org/10.1145/1142980.1142982