Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

A Contract Framework for Reasoning about Safety and Progress

Imene Ben-Hafaiedh, Susanne Graf and Sophie Quinton

Verimag Research Report n^o TR-2010-11

14-05-2010

A Contract Framework for Reasoning about Safety and Progress

Imene Ben-Hafaiedh, Susanne Graf and Sophie Quinton

14-05-2010

Abstract

Designing concurrent or distributed systems with complex architectures while preserving a set of high-level requirements through all design steps is not a trivial task. An approach which is both compositional and incremental is mandatory to master this complexity. We propose here a contract-based design and verification framework for safety and progress requirements in component systems with data exchange. We build upon a notion of *contract framework* which relies on a *component algebra* and two refinement relations: *conformance* and *refinement under context*. We provide a condition under which circular reasoning can be used for checking *dominance*, i.e. refinement between contracts. We illustrate our verification methodology with a case study: a protocol for tree-like networks for which both safety and progress must be ensured.

Keywords: contracts, component-based systems, incremental verification, compositional verification

Reviewers:

Notes:

How to cite this report:

@techreport { ,
title = { A Contract Framework for Reasoning
about Safety and Progress},
authors = { Imene Ben-Hafaiedh, Susanne Graf and Sophie Quinton},
institution = { Verimag Research Report },
number = {TR-2010-11},
year = { },
note = { }
}

1 Introduction

We aim at the definition of a scalable design and verification methodology for rich specifications of distributed component systems of arbitrary size expressing safety and progress properties. For expressing such rich, yet abstract specifications, we propose a formalism similar to symbolic transition systems as introduced in [10] which we extend in several ways. We define progress constraints generalizing the usual strong and weak fairness and we decorate control states with invariants on state variables. We also consider an explicit composition model represented by sets of *connectors*. Each connector defines a set of interactions and a transformation on (non persistent) port variables, where *ports* name transitions of the local components involved in the interaction. For achieving scalability, we base verification on an abstract semantics in which explicit values of state variables are abstracted by the defined state invariants.

For achieving independent implementation of such specifications, we propose to use *contracts*. Like in contract-based design [11], we use contracts to constrain, reuse and replace implementations, and not to support assume/guarantee based compositional verification — where assumptions are used to deduce global properties φ (see [9]). Contracts are design constraints for implementations which are maintained throughout the development and life cycle of the system. As we are interested in system rather than program design, we consider expressive contracts specifying temporal safety and progress properties rather than pre- and post conditions.

Interfaces [8] have been proposed for this purpose. Here, we consider contracts of the form (A, gl, G) where gl is a composition operator. The reason is that we are interested in rich exogenous composition operators which allow to represent abstractions of protocols, middleware components and orchestrations whereas assumptions and guarantees constrain peers at the same layer. Some formalisms for describing such rich connectors abstractly have been proposed, e.g., the Kell calculus [4] or the connector calculus Reo [1]. Kell is, however, mainly concerned with obtaining correctly typed connectors, and Reo supposes independence amongst connectors and does not take into account constraints imposed by components. We choose composition operators defined in terms of rich connectors of the BIP component framework [2] which have the required expressiveness, define interactions with component behaviors and allow handling conflicting connectors.

As in [12], we promote here a two-phase approach for defining a contract framework which has not been done before at this level of generality: we first define a general notion of contract framework stating the necessary ingredients — a component framework, notions of conformance (for ensuring global properties φ), satisfaction (of contracts by implementations), and dominance (refinement between contracts). We provide interesting rules for establishing dominance and validity conditions for them. In particular, the loose coupling between satisfaction and conformance may allow a powerful rule based on so-called *circular reasoning*. For any particular contract framework, we have then only to establish the required validity conditions. We propose here 2 improvements with respect to [12]: (a) we do not suppose a fixed composition framework but composition is a parameter required to satisfy a number of general properties — in particular flattening and structuring — and we extend the framework to take into account port hiding which is a key ingredient for proving refinement between specifications at different levels of granularity.

1.0.1 Organization.

Section 2.0.3 presents the overall design and verification methodology we are aiming at. We introduce and extend the relevant features of the contract framework of [12]. Section 3 defines a particular contract framework for a component framework allowing the expression of safety and liveness also on state and port variables, and the already mentioned rich exogenous composition. Finally, Section 4 applies the methodology to a resource sharing algorithm in a networked system of arbitrary size, where all necessary verification

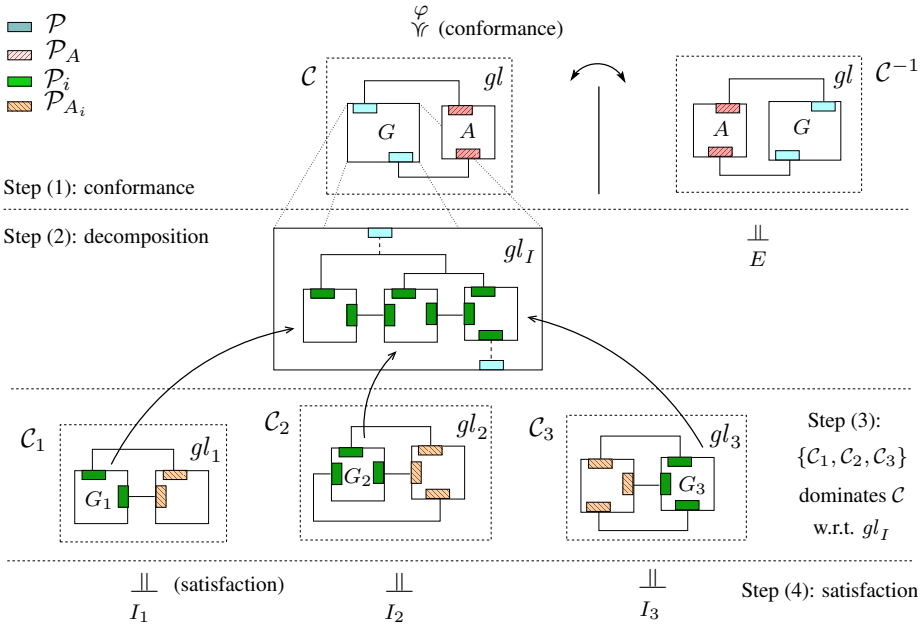


Figure 1: Methodology steps ensuring that $gl\{E, gl_I\{I_1, I_2, I_3\}\} \preccurlyeq \varphi$

steps are automated in a tool.

2 A contract-based design framework and methodology

We first explain the contract-based design methodology that we target. Then we define a generic notion of contract framework supporting this methodology.

2.0.2 Design methodology.

We represent our design and verification methodology in a top-down fashion in which high-level properties are pushed progressively from the overall system into atomic components — which we call implementations. As usually, this is just a convenient representation; in real life, we will always achieve the final picture in several iterations alternatively going up and down. We are interested in systems with a complex architecture which are potentially of arbitrary size. Figure 1 illustrates this methodology.

We suppose given a global property φ which the system K under construction has to realize together with an environment on which we may have some knowledge, expressed by a property A. φ and A are expressed w.r.t the interface $\mathcal{P}_K$ of K. We proceed as follows: (1) define a *contract* $\mathcal{C}$ for $\mathcal{P}_K$ which *conforms* to φ; (2) define K as a composition of subcomponents K_i and a contract $\mathcal{C}_i$ for each of them; possibly iterate this step if needed. (3) prove that any set of implementations (components) for K_i *satisfying* the contracts $\mathcal{C}_i$, when composed, satisfies the top-level contract $\mathcal{C}$ (*dominance*) — and thus guarantees φ; (4) provide such implementations.

The global property φ appears at the top, while the implementations I_i are at the bottom. An additional step is represented on the right hand side: it allows the integration of K in an actual environment E while preserving φ. For this, E must satisfy the "mirror" contract[1] of $\mathcal{C}$.

The correctness proof for a particular system is split into 3 phases: *conformance* of the top-level contrat $\mathcal{C}$ to φ, *dominance* between the contracts $\mathcal{C}_i$ and $\mathcal{C}$, *satisfaction* of the $\mathcal{C}_i$ by the implementation I_i. Section 2 introduces the notion of contract framework and properties that *conformance*, *dominance* and *satisfaction* must ensure in order to support this methodology. In section 3, we present an actual contract framework based on symbolic transition systems and rich connectors, which is expressive enough for the properties we want to prove and has the properties required by the general framework. Finally, for a given application presented in section 4, we do the actual conformance, dominance and satisfaction proofs with the help of a tool developed for this purpose.

A *context* for an interface $\mathcal{P}$ describes how a component with interface $\mathcal{P}$ is intended to be connected to its environment E and provides a property expected from E. In the sequel, we denote composition operators by gl — standing for *"glue"* [13]. A context is then of the form (gl, A). A *contract* for an interface $\mathcal{P}$ consists of a context (gl, A) and a property G on $\mathcal{P}$ that the component under design must ensure in the given context in order to *satisfy* this contract. *Conformance* relates properties of closed systems and *dominance* relates contracts.

2.0.3 Contract frameworks.

The methodology we propose relies on a framework supporting hierarchical components as well as some powerful mechanisms to reason about composition. We therefore introduce next the notions a component framework must define and the properties it has to satisfy — this is inspired by [13].

Definition 2.1 (Component algebra) *A* component algebra *is a structure of the form* $(\mathcal{K}, GL, \circ, \cong)$ *where:*

- $\mathcal{K}$ *is a set of* components *— describing their* behavior *or* properties.
  Each component $K \in \mathcal{K}$ *has as its* interface *a set of* ports*, denoted* $\mathcal{P}_K$.
- GL *is a set of* glue (composition) operators.
  Operators $gl \in GL$ *are partial functions* $2^{\mathcal{K}} \longrightarrow \mathcal{K}$ *transforming a set of components into a new component. Each* gl *is defined on a set of ports* P_{gl} *— of the original set of components, called its* support set *— and defines a new interface* $\mathcal{P}_{gl}$ *— on the new component, called its* exported interface*. Thus,* $K = gl\{K_1, \ldots, K_n\}$[2] *is defined if* $K_1, \ldots, K_n \in \mathcal{K}$ *have disjoint interfaces,* $P_{gl} = \bigcup_{i=1}^{n} \mathcal{P}_{K_i}$ *and the interface of* K *is* $\mathcal{P}_{gl}$*, the exported interface of* gl.
- $\circ$ *is an operation on* GL *allowing to compose glues. It is such that* $(GL, \circ)$ *is a commutative monoid.*[3]
- $\cong \subseteq \mathcal{K} \times \mathcal{K}$ *is an equivalence relation*[4].

Composition operators allow building hierarchical components from atomic ones. As explained in [13], two types of transformations of components are of interest when dealing with a component algebra (see Figure 2). A usual one is *flattening*, which allows to transform an arbitrary hierarchical component into a flat one consisting of a unique glue composing all the components: consider 3 operators gl_1 and gl_2 defined on

[1] or any contract dominating this mirror contract.

[2] For simplicity of notation, we denote $gl(\{K_1, \ldots, K_n\})$ by $gl\{K_1, \ldots, K_n\}$.

[3] Formally, $gl \circ gl'$ is defined on $(P_{gl} \cap P_{gl'}) \cup (P_{gl} \backslash \mathcal{P}_{gl'}) \cup (P_{gl'} \backslash \mathcal{P}_{gl})$ and defines as interface $(\mathcal{P}_{gl} \cup \mathcal{P}_{gl'}) \backslash P_{gl \circ gl'}$. Note that $gl \circ gl'$ must be defined even if $(P_{gl} \cap P_{gl'}) \neq \emptyset$.

[4] In general, this equivalence is derived from equality or equivalence of semantic sets.

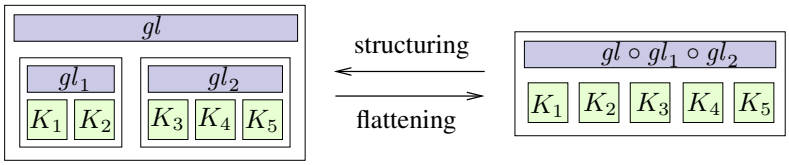


Figure 2: Structuring and flattening of a component

respectively P_1 and P_2, and gl defined on $P = \mathcal{P}_{gl_1} \cup \mathcal{P}_{gl_2}$. Then, we require that $gl\{gl_1\{\mathcal{K}_1\}, gl_2\{\mathcal{K}_2\}\} \cong (gl \circ gl_1 \circ gl_2)\{\mathcal{K}_1 \cup \mathcal{K}_2\}$ for any sets of components $\mathcal{K}_i$ such that all terms are defined.

The second transformation is *structuring*; it goes in the opposite direction and allows to decompose a component according to any partition of its subcomponents: for any gl defined on $P = P_1 \cup P_2$, there must exist gl', gl_1 and gl_2 such that $gl\{\mathcal{K}_1 \cup \mathcal{K}_2\} \cong gl'\{gl_1\{\mathcal{K}_1\}, gl_2\{\mathcal{K}_2\}\}$ for any $\mathcal{K}_i$ such that all terms are defined. While most usual component frameworks allow flattening, structuring is more difficult to achieve. It is needed in order to build the context of a component K in a given system — which is necessary to verify that the assumption of contract $\mathcal{C}_K$ is discharged.

We now formally define the notion of *context* limiting the way in which a component may be further composed.

Definition 2.2 (Context) *A* context *for an interface $\mathcal{P}$ is a pair (E, gl) where E is such that $\mathcal{P} \cap \mathcal{P}_E = \emptyset$ and gl is defined on $\mathcal{P} \cup \mathcal{P}_E$.*

As promised, we introduce two refinement relations to reason about contracts: *conformance* has been mentioned in the methodology; the second one, called *refinement under context*, is used to define *satisfaction* and *dominance*. Note that refinement under context is usually considered as a derived relation and chosen as the weakest relation implying conformance ensuring compositionality, i.e. preservation by composition. We allow a looser coupling between these two refinements so as to obtain stronger reasoning schemata for dominance.

Definition 2.3 (Contract framework) *A* contract framework *is a tuple $(\mathcal{K}, GL, \circ, \cong, \{\sqsubseteq_{E,gl}\}, \preccurlyeq)$ where:*

- *$(\mathcal{K}, GL, \circ, \cong)$ is a component algebra allowing flattening and structuring*
- *$\{\sqsubseteq_{E,gl}\}$ is a* refinement under context *relation parameterized by a context. Given a context (E, gl) for an interface $\mathcal{P}$, $\sqsubseteq_{E,gl}$ is a preorder over the set of components on $\mathcal{P}$ which is expected to be compositional.*
- *$\preccurlyeq \subseteq \mathcal{K} \times \mathcal{K}$ is a* conformance *relation relating components with the same interface. It is a preorder such that for any K_1, K_2 on the same interface $\mathcal{P}$ and for any context (E, gl) for $\mathcal{P}$, $K_1 \sqsubseteq_{E,gl} K_2 \implies gl\{K_1, E\} \preccurlyeq gl\{K_2, E\}$.*

Definition 2.4 (Contract) *A contract $\mathcal{C}$ for an interface $\mathcal{P}$ consists of:*

- *a context $\mathcal{E} = (A, gl)$ for $\mathcal{P}$ where A is called the* assumption
- *a component G on $\mathcal{P}$ called the* guarantee

A component K satisfies *a contract $\mathcal{C}$, denoted $K \models \mathcal{C}$, iff $K \sqsubseteq_{A,gl} G$.*

We write $\mathcal{C} = (A, gl, G)$ rather than $\mathcal{C} = ((A, gl), G)$. gl implicitly defines the interface of the environment while A expresses a constraint on it and G a constraint on the refinements of K. The "mirror" contract $\mathcal{C}^{-1}$ of $\mathcal{C}$ is (G, gl, A).

In interface theories [8], a single constraint allows to derive both A and G (gl is predefined), because each transition is controlled either by the component or the environment. However, in frameworks with rendez-vous interaction, several pairs (A,G) may be derived. Thus, we keep assumptions and guarantees separate.

Dominance is the key notion that distinguishes reasoning in a contract or interface framework from theories based on refinement between components. Contract $\mathcal{C}$ is said to dominate contract $\mathcal{C}'$ if every implementation of $\mathcal{C}$ — i.e., every component satisfying $\mathcal{C}$ — is also an implementation of $\mathcal{C}'$. Intuitively, this is achieved by a $\mathcal{C}'$ that has a stronger promise or a weaker assumption than $\mathcal{C}$.

In our general setting — which does not refer to any particular composition or component model — it is not sufficient to define dominance just on a pair of contracts. A typical situation that we have to handle is that of a hierarchical component depicted in Figure 1, where a set of contracts $\{\mathcal{C}_i\}_{i=1}^n$ is defined for the inner components (on disjoint interfaces $\{\mathcal{P}_i\}_{i=1}^n$) and a contract $\mathcal{C}$ for the hierarchical component whose interface is the exported interface of a composition operator gl_I defined on $P = \bigcup_{i=1}^n \mathcal{P}_i$. It looks attractive to solve such a dominance problem by defining a contract algebra as in [3], as checking dominance boils then down to checking whether $\widetilde{gl}\{\mathcal{C}_1, \dots, \mathcal{C}_n\}$ dominates $\mathcal{C}$ for some operator $\widetilde{gl}$ on contracts. This is, however, not possible for arbitrary component frameworks. We thus provide a broader dominance defined directly for a set of contracts $\{\mathcal{C}_i\}_{i=1}^n$ and a contract $\mathcal{C}$ to be dominated w.r.t a composition operator gl_I.

In order to allow hiding ports of the lower-level contracts which do not appear at the interface of the top-level contract, we relax the constraints on the composition operators by only requiring that they agree on their common ports. For this, we need a notion of *projection* of a component K onto a subset P' of its interface, denoted $\Pi_{P'}(K)$, which is quite natural and must preserve some properties detailed in Appendix A. Hence the following semantic definition of dominance.

Definition 2.5 (Dominance) *$\{\mathcal{C}_i\}_{i=1}^n$ dominates $\mathcal{C}$ w.r.t. gl_I iff:*

- *for every i, there exists a glue gl_{E_i} s.t. $gl \circ gl_I = gl_i \circ gl_{E_i}$*
- *for any components $\{K_i\}_{i=1}^n$, $(\forall i, K_i \models \mathcal{C}_i) \implies \Pi_P(gl_I\{K_1, \dots, K_n\}) \models \mathcal{C}$*

A sufficient condition for dominance has been proposed in [12], without the relaxation condition on composition operators. The generalization of this condition is given below. It relies on the fact that local assumptions are indeed discharged, that is, implied by the environment defined by the guarantees of the peers and the global assumption A. It requires a specific property called soundness of circular reasoning, which ensures that a set of peer contracts can be used to mutually discharge their assumptions, and it is formally expressed by: $K \sqsubseteq_{A,gl} G \wedge E \sqsubseteq_{G,gl} A \implies K \sqsubseteq_{E,gl} G$. More detail is given in Appendix B.

Theorem 2.6 *If circular reasoning is sound and $\forall i.\ \exists gl_{E_i}.\ gl \circ gl_I = gl_i \circ gl_{E_i}$, then to prove that $\mathcal{C}$ dominates $\{\mathcal{C}_i\}_{i=1..n}$ w.r.t. gl, it is sufficient to prove that:*

$$\begin{cases} \Pi_P(gl_I\{G_1, \dots, G_n\}) \models \mathcal{C} \\ \forall i, \Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, G_1, \dots, G_{i-1}, G_{i+1}, \dots, G_n\}) \models \mathcal{C}_i^{-1} \end{cases}$$

This shows that the proof of a dominance relation boils down to a set of refinement checks, one for proving refinement between the guarantees, the second for discharging individual assumptions. A proof is given in Appendix C.

2.0.4 Methodology.

We now extend our design and verification methodology to recursively defined systems so that we can handle systems representing component networks of arbitrary size defined by a component grammar as follows:

- a set of terminal symbols $\{E, I_1, \dots, I_k\}$ representing implementations;
- a set of nonterminal symbols $\{S, K_0, K_1, \dots, K_n\}$ representing hierarchical components; S, which defines the top-level closed system, is the axiom;
- a set of rules corresponding to design steps which define each non-terminal either as a composition of subsystems or as an implementation:
  - $S \longrightarrow gl\{E, K_0\}$.
  - For $i \in [0, n]$, at least one rule either of the form $K_i \longrightarrow I_j$ ($j \in [1, k]$) or $K_i \longrightarrow gl_{\Sigma_i}\{K_j\}_{j\in\Sigma_i}$, where Σ_i a set of indices and gl_{Σ_i} a composition operator on the union of the interfaces of the K_j.

Unlike classical network grammars, we use "rich" composition operators and are not limited to flat regular networks, as for example in [14]. We now instantiate the methodology of Figure 1 for such component networks. We choose again a top-down presentation, but it is possible to proceed in a different order.

1. formulate a top-level requirement φ characterizing the closed system defined by the system and its environment
2. define a contract $\mathcal{C} = (A, gl, G)$ associated with K_0 and prove that $gl\{A, G\} \preccurlyeq \varphi$
3. prove (or justify informally if this is not possible) that the actual environment E satisfies the "mirror" contract of $\mathcal{C}$: $E \models (G, gl, A)$
4. define for every non terminal K_i a contract $\mathcal{C}_{K_i} = (A_{K_i}, gl_{K_i}, G_{K_i})$ such that for every rule $K_l \longrightarrow gl_{\Sigma_l}\{K_j\}_{j\in\Sigma_l}$ having an occurrence of K_i on the right hand side, gl_{K_i} is compatible with $gl_{\Sigma_l} \circ gl_{K_l}$
5. for each $K_i \longrightarrow gl_{\Sigma_i}\{K_j\}_{j\in\Sigma_i}$, show that $\{\mathcal{C}_{K_j}\}_{j\in\Sigma_i}$ dominates $\mathcal{C}_{K_i}$ w.r.t gl_{Σ_i}
6. prove that implementations satisfy their contract: $K_i \longrightarrow I_j \implies I_j \models \mathcal{C}_{K_i}$

Theorem 2.7 *Let $\mathcal{G}$ be a grammar such that all methodology steps have been completed to guarantee a requirement φ. Any component system corresponding to a word accepted by $\mathcal{G}$ satisfies φ.*

The proof is a simple induction on the number of steps required for deriving the component from S, showing that conformance is preserved from the left-hand side to the right-hand side of a rule. If φ can express progress and if dominance preserves progress, this methodology is sufficient for systems with a unique requirement but also for multiple requirements decomposed according to the same network grammar.

3 A contract framework with data for safety and progress

In this section, we define a contract framework in order to prove safety and progress properties of distributed systems. We choose to use composition operators based on the BIP interaction model [13, 5] because of their expressiveness and their properties making them suitable for structural verification. Our framework handles variables, guards and data transfer — which are supported by the BIP interaction model [6] — and furthermore is adequate for loose specifications.

3.0.5 Components.

A component is defined by a labeled transition system enriched with variables. In order to allow abstract descriptions of components, we handle predicates on variables rather than concrete values. Except for τ, which denotes internal actions, labels are ports of the component interface. For example, a transition t labeled by port p denotes that the component will perform the action associated with t only if an interaction in which p is involved happens. Our components also provide some progress properties which are described below.

A port p is sometimes represented along with its associated variables $x_1, \dots, x_n$, which is denoted $p[x_1, \dots, x_n]$. Without loss of generality, we suppose in the following that a port is associated with exactly one variable. We suppose given a set of predicates that is closed by $\wedge$ and $\vee$.

Definition 3.1 (Component) *A* component *is a tuple* $(TS, X, I, g, f, Prog)$*:*

- $TS = (Q, q^0, P \cup \{\tau\}, \longrightarrow)$ *is a labeled transition system:* Q *is a set of* states*,* $q^0 \in Q$ *is the* initial state*,* $P \cup \{\tau\}$ *is a set of labels.* $\longrightarrow \subseteq Q \times P \cup \{\tau\} \times Q$ *is a* transition *relation. Elements of* P *are* ports *and* τ *labels* internal *transitions.*
  As usual, a transition $(q, p, q') \in \longrightarrow$ *is denoted* $q \xrightarrow{p} q'$*;*
- X *is a set of* variables*. Some variables are associated with a (unique) port;*
  $X^{st} \subseteq X$ *contains* state variables *which are denoted* $st_1, \dots, st_s$*. Relation* R *relates*[5] *variables in* X *to variables in* X^{st}*;*
- I *associates with every* $q \in Q$ *a* state invariant I_q *that is a predicate on* X^{st}*;*
- g *associates with every transition* t *a* guard g_t*, i.e. a predicate on* X^{st}*;*
- f *associates with every transition* t *an* action f_t *defined as a predicate on* $X^{st} \cup \{x_\gamma\} \cup X^{st}_{new}$ *where* x_γ *is the variable associated with the port labeling* t[6] *and* $X^{st}_{new} = \{st_1^{new}, \dots, st_s^{new}\}$ *represents the "updated" variables;*
- $Prog$ *a set of* progress properties *(see below).*

3.0.6 Progress properties.

When considering abstract specifications, progress properties are useful to exclude behaviors staying forever in some particular states or loops. We adapt usual weak and strong fairness conditions to component systems: a *progress property* $pr \in Prog$ for a component K is a pair of transition sets (T_c, T_p), where T_c is called the *condition* and T_p the *promise*. We define the set of *progress states* of T_p, denoted $start(T_p)$, as the set of initial states of transitions of T_p.

(T_c, T_p) is a valid progress property iff: considering an execution σ of K in some context containing infinitely many T_c-transitions, in every state of $start(T_p)$ occurring infinitely often, at least one transition of T_p appears infinitely often in σ, unless the environment forbids it. $(\top, T_c)$ denotes *unconditional progress*, which means that σ cannot stay forever in $start(T_p)$ without firing infinitely often a transition of T_p.

Note that (T_c, T_p) is trivially satisfied if no T_c-transition can be fired infinitely often. When T_p is empty or not reachable from any "T_c-loop", (T_c, T_p) is a progress property only if no T_c-transition can be fired infinitely often. Monotonicity properties w.r.t. progress which allow inferring new progress properties from existing ones are given in Appendix D.

[5] Non-state variables are transient. R produces their value whenever it is necessary.

[6] If there is no associated variable (t is labeled by p_γ with $\gamma \in \mathcal{I}_{obs}$ or by τ), f_t is a predicate on $X^{st} \cup X^{st}_{new}$.

3.0.7 Semantics.

The concrete semantics of a component is the usual SOS semantics for labeled transition systems. We do not need it in the following because we only work with an abstract semantics of components: the latter is a labeled transition system in which there exists a transition iff there exists a concrete valuation of the variables for which the transition can be fired. Our semantics is a *closed* semantics, because we suppose that the environment of the component does not affect the values of the variables attached to ports labeling transitions. This strongly motivates a design framework using contracts, that defines closed systems.

Definition 3.2 (Abstract semantics) *Let $K = (TS, X, g, f, I, Prog)$ be a component. The abstract semantics of K is the transition system $(Q, q^0, P, \hookrightarrow)$ where $q \overset{p[x]}{\hookrightarrow} q'$ iff there exist a transition $t = (q \overset{p[x]}{\longrightarrow} q')$ such that the predicate $(st_1, ..., st_s)\, R\, x \wedge Sem_t$ is satisfiable, where Sem_t denotes $I_q \wedge g_t \wedge f_t \wedge I_{q'}$.*

Note that a transition $t = (q \overset{p}{\longrightarrow} q')$ is not preserved in the semantics if f_t is not consistent with $I_{q'}$ — meaning that firing t leads to a state in which $I_{q'}$ cannot hold. Thus, in order to avoid deadlocks in the semantics, the state invariants must respect some consistency and completeness conditions.

3.0.8 Composition.

We now define the composition operators that allow us to build complex components based on atomic ones. These composition operators are called *interaction models* and they are made of *connectors*.

From the possible synchronizations offered by the BIP framework (see [5]), we keep only two basic types of connectors: *rendez-vous* connectors require *all* ports to be activated in order for the interaction to take place and involve data transfers; interactions in an *observation* connector can take place as soon as *any* port is activated, and no data is exchanged. Thus, adding observation connectors does not modify the behavior of the system, hence their name. Two (or more) connectors of the same type can be composed to build a *hierarchical* connector simply by using the exported port of one connector as an element of the support set of the other.

Definition 3.3 (Rendez-vous connector) *A* rendez-vous *connector $\gamma = (p[x], P, \delta)$ is defined by:*

- *$p[x]$, the* exported port *and $P = \{p_1[x_1], ..., p_k[x_k]\}$, the* support set *of ports*
- *$\delta = (G, \mathcal{U}, \mathcal{D})$ where:*
  - *G is the* guard*, that is, a predicate on $X = \{x_1, ..., x_k\}$*
  - *$\mathcal{U}$ is the* upward update function *defined as a predicate on $X \cup \{x\}$*
  - *For $x_i \in X$, $\mathcal{D}_{x_i}$ is a* downward update function*, i.e., a predicate on $\{x\} \cup \{x_i\}$*

where $\mathcal{D}_{x_i}$ is the function that returns the projection of $\mathcal{D}$ corresponding to x_i. Note that $\mathcal{U}$ and $\mathcal{D}$ must be associative to allow structuring *(see figure 2).*

As *observation* connectors do not involve data transfer, they have neither guard nor $\mathcal{U}$ nor $\mathcal{D}$ predicates. The variables attached to ports are useless and thus hidden. Hence the following definition.

Definition 3.4 (Observation connector) *An* observation *connector $\gamma = (p, P)$ is defined by an* exported port *p and a* support set *$P = \{p_1, ..., p_k\}$.*

To avoid cyclic connectors, we require also that $p \notin P$. Two connectors γ_1 and γ_2 are *disjoint* if $p_1 \neq p_2$, $p_1 \notin P_2$ and $p_2 \notin P_1$. Note that P_1 and P_2 may have ports in common, as a port may be connected to several connectors.

We can now define our composition operators as sets of connectors.

Definition 3.5 (Interaction model) *An interaction model $\mathcal{I}$ defined on a set of ports P is a set of disjoint connectors $\gamma_i = (p_i[x], P_i, \delta_i)$ where $P = \bigcup_{i=1}^{n} P_i$. We denote by $\mathcal{I}_{rdv}$ the set of rendez-vous connectors of $\mathcal{I}$ and $\mathcal{I}_{obs}$ the set of its observation connectors.*

We associate with an interaction model $\mathcal{I}$ an interface $\mathcal{P}_\mathcal{I}$ consisting of the set of the exported ports of its connectors. This means that the interface of the component resulting from a composition using $\mathcal{I}$ has only these exported ports as labels. $X_\mathcal{I}$ denotes the set of variables associated with the ports of $\mathcal{P}_\mathcal{I}$.

Merge of connectors is the operation that takes two connectors defining together a hierarchical connector and returns a connector of a basic type. Merge is defined for rendez-vous connectors in [6] (where it is called flattening). We restrict this definition so as to preserve associativity of the upward and downward functions. Merge of observation connectors has been described in [5]. These definitions extend naturally to our interaction models, where rendez-vous and observation connectors are merged separately (see Appendix E).

We now define composition: given a set of components $K_1, \dots, K_n$ and an interaction model $\mathcal{I}$, we build a compound component denoted $\mathcal{I}\{K_1, \dots, K_n\}$, with $\mathcal{P}_\mathcal{I}$ as interface. As we do not allow sets of ports as labels of transitions, we require that connectors of $\mathcal{I}$ have at most one port of the same component in their support set. Composition is rather technical but straightforward. It does not involve hiding of ports. Besides, a variable of $\mathcal{I}\{K_1, \dots, K_n\}$ is a variable of some K_i or a variable associated with the exported port of some $p_\gamma \in \mathcal{I}$.

Definition 3.6 (Composition of components) *Let $\{P_i\}_{i=1}^{n}$ be a family of pairwise disjoint interfaces and $P = \bigcup_{i=1}^{n} P_i$. Let $\mathcal{I}$ be an interaction model on P. For $i \in [1, n]$, let $K_i = (TS_i, X_i, g_i, f_i, I_i, Prog_i)$ be a component on P_i. The* composition *of $K_1, \dots, K_n$* with *$\mathcal{I}$ is a component $(TS, X, g, f, I, Prog)$ such that:*

- *$TS = (Q, q^0, \mathcal{P}_\mathcal{I} \cup \{\tau\}, \longrightarrow)$ with $Q = \prod_{i=1}^{n} Q_i$, $q^0 = (q_1^0, \dots, q_n^0)$ and where $\longrightarrow$ is the least set of transitions satisfying the following rules[7]:*

$$\frac{(p_\gamma, P_\gamma, \delta_\gamma) \in \mathcal{I}_{rdv} \quad \forall i \in [1,n].\ q_i \xrightarrow{P_i \cap P_\gamma}_i q_i'}{(q_1, \dots, q_n) \xrightarrow{p_\gamma} (q_1', \dots, q_n')} \qquad \frac{\exists i \in [1,n].\ q_i \xrightarrow{\tau}_i q_i'}{(q_1, \dots, q_n) \xrightarrow{\tau} (q_1, \dots, q_i', \dots, q_n)}$$

  with the convention that $q_i \xrightarrow{\emptyset}_i q_i'$ iff $q_i = q_i'$. Note that $|P_i \cap P_\gamma| \leq 1$.

- *$X^{st} = \bigcup_{i=1}^{n} X_i^{st}$ and $X = \bigcup_{i=1}^{n} X_i \cup X_\mathcal{I}$*
  The relation R between variables in X and state variables is defined as:
  Case 1: $x \in X_i$ for some $i \in [1, n]$. $x\, R\, (st_1, \dots, st_s)$ iff $x\, R_i\, (st_1^i, \dots, st_{s_i}^i)$, where $\{st_1^i, \dots, st_{s_i}^i\} = X_i^{st} \subseteq X^{st}$.
  Case 2: $x \in X_\mathcal{I}$. Then x is associated with the exported port p_γ of a rendez-vous connector $\gamma = (p_\gamma, P_\gamma, \delta) \in \mathcal{I}_{rdv}$. Let $k = |P_\gamma|$. $\mathcal{U}_\gamma$ is a predicate on $\{x_1, \dots, x_k\} \cup \{x\}$, where every x_i is associated with a port of P_γ. Without loss of generality, we suppose each x_i is a variable of component K_i. Then $x\, R\, (st_1, \dots, st_s)$ is defined iff:

$$\exists v_1, \dots, v_k.\ (\forall i \in [1,k].\ v_i\, R_i\, (st_1^i, \dots, st_{s_i}^i)) \wedge \mathcal{U}_\gamma[x_1/v_1, \dots, x_k/v_k]$$

  where $\mathcal{U}_\gamma[x_1/v_1, \dots, x_k/v_k]$ is the predicate on x obtained by replacing the variables $x_1, \dots, x_k$ by values $v_1, \dots, v_k$ compatible with the local relations R_i between the x_i and the local state variables.

[7] The rule for connectors in $\mathcal{I}_{obs}$ is similar to the one for rendez-vous connectors except that any subset of the support set P_γ may participate in the interaction.

- *For each* $q \in Q$, $I_q = \bigwedge_{\{i=1\}}^{n} I_{q_i}$
- *Consider* $t = (q_1, \dots, q_n) \xrightarrow{p_\gamma} (q'_1, \dots, q'_n)$ *for* $\gamma \in \mathcal{I}_{rdv}$[8]. *W.l.o.g., we suppose* $P_\gamma = \{x_1, \dots, x_k\}$ *with* $x_i \in P_i$ *for every* i *in* $[1, k]$. *For* $i \in [1, k]$, *the local transition* $(q_i \xrightarrow{p_i[x_i]}_i q'_i)$ *corresponding to* t *is denoted* $\pi_i(t)$. *Again,* $\{st^i_1, \dots, st^i_{s_i}\} = X^{st}_i \subseteq X^{st}$.
  - $g_t(st_1, \dots, st_s)$ *holds iff the following holds:*
    - $\forall i \in [1, k].\ g_{t_i}(st^i_1, \dots, st^i_{s_i})$
    - $\exists v_1, \dots, v_k.\ (\forall i \in [1, k].\ v_i\, R_i\, (st^i_1, \dots, st^i_{s_i})) \wedge G[x_1/v_1, \dots, x_k/v_k]$
  - $f_t(st_1, \dots, st_s, x_\gamma, st^{new}_1, \dots, st^{new}_s)$ *holds iff* $\exists v_1, \dots, v_k$ *s.t. it holds that:*
    - $\mathcal{D}[x_1/v_1, \dots, x_k/v_k]$, *which is a predicate on* x_γ
    - $\forall i \in [1, k].\ f_{\pi_i(t)}[x_i/v_i]$, *which is a predicate on* $X^{st}_i \cup X^{st}_{i,new}$
- *Prog is defined below (see definition 3.7)*

We never explicitly construct all (strongest) progress properties for a composition: compositions are only built as far as needed to prove dominance. Thus, we only give below a condition for checking that a pair of sets (T_c, T_p) is a progress property of a composition by checking that the projections of (T_c, T_p) onto individual components are local progress properties.

Definition 3.7 (Progress property in a composition) (T_c, T_p) *is a progress property of* $\mathcal{I}(K_1, \ ..., \ K_n)$ *if* $\forall i \in [1, n]$*:*

- *either* $\pi_i(T_p)$ *never contains more than one joint transition of* $\mathcal{I}$ *from the same state and then* $(\pi_i(T_c), \pi_i(T_p))$ *is a local progress property.*
- *or it does, and then we split* $\pi_i(T_p)$ *into a set of promises* $T^{i,1}_p \dots, T^{i,k}_p$ *containing exactly one joint transition for each state before checking that all pairs in* $\{(T^i_c, T^{i,1}_p), \dots, (T^i_c, T^{i,k}_p)\}$ *are local progress properties*[9].

3.0.9 Refinement.

Refinement under context ensures that in the given context $(E, \mathcal{I})$ — and in any context refining it — safety and progress properties are preserved from the abstract component K_{abs} to the refined component K_{conc}. Moreover, refinement under context allows circular reasoning for the considered composition operators (provided that the assumptions are deterministic), because the enabledness of transitions must be preserved from K_{abs} to K_{conc} in any state reachable in the considered context. To simplify the definition, we suppose that (a) K_{abs} has no internal transitions, (b) E has no transitions that it may do alone and (c) progress is refined without taking into account the context. The first two steps imply no loss of generality. The last simplification is sufficient for the considered application. It could be refined by requiring from K_{conc} only (part of) the progress conditions of K_{conc} which are meaningful in $(E, \mathcal{I})$.

Refinement is defined by means of two relations (1) α relating variables of K_{conc} and K_{abs}, and (2) $\mathcal{R}$ relating concrete and abstract states. For preserving progress, we project transition sets of K_{abs} onto K_{conc} — for this purpose, we define the following auxiliary notations.

[8] For a transition labeled by p_γ with $\gamma \in \mathcal{I}_{obs}$, only the conditions on the local guard and function of the components involved in the interaction are kept. The guard and function of a τ-transition are the corresponding local guard and function.

[9] This is necessary to avoid that different processes choose a different joint transition in a given initial state.

Definition 3.8 (Projection) *Let $\mathcal{R}$ be a relation on $(Q_{conc} \times Q_E) \times Q_{abs}$. We define the projection $\overline{\mathcal{R}}$ of $\mathcal{R}$ onto $Q_{conc} \times Q_{abs}$ by $q_c \,\overline{\mathcal{R}}\, q_a$ iff $\exists q_E$ s.t. $(q_c, q_E)\,\mathcal{R}\, q_a$. For $Q_a \subseteq Q_{abs}$, we denote $\overline{\mathcal{R}}^{-1}(Q_a) \subseteq Q_{conc}$ the inverse image of Q_a under $\overline{\mathcal{R}}$. $\overline{\mathcal{R}}^{-1}(\{q_a \overset{p}{\hookrightarrow}_{abs} q'_a\})$ denotes the set of p-transitions of K_{conc} between states in $\overline{\mathcal{R}}^{-1}(\{q_a\})$ and in $\overline{\mathcal{R}}^{-1}(\{q'_a\})$. This notation extends naturally to transition sets.*

Definition 3.9 (Refinement under context) *Given a relation α on $X_{conc} \cup X_{abs}$, K_{conc} refines K_{abs} in the context of $(E, \mathcal{I})$, denoted $K_{conc} \sqsubseteq_{E,\mathcal{I}} K_{abs}$, iff:*

(a) $\exists \mathcal{R} \subseteq (Q_{conc} \times Q_E) \times Q_{abs}$ s.t. $(q_c^0, q_E^0)\,\mathcal{R}\, q_a^0$ and s.t. $(q_c, q_E)\,\mathcal{R}\, q_a$ implies:

1. $I_{q_c} \wedge \alpha(X_{conc}, X_{abs}) \implies I_{q_a}$
2. *$\forall p[x] \in P$, the following holds ($\mathcal{V}_i$ denotes a valuation of X_i):*
   - *for any value v of x: $\exists t_c = q_c \xrightarrow{p}_c q'_c$ and $\mathcal{V}_c, \mathcal{V}_c^{new}$ satisfying Sem_{t_c} implies $\exists q'_a, t_a = q_a \xrightarrow{p}_a q'_a$ and $\mathcal{V}_a, \mathcal{V}_a^{new}$ consistent with α and satisfying Sem_{t_a}.*
   - *$\exists \gamma.\ P_\gamma = \{p, e\} \wedge (q_c, q_E) \overset{p_\gamma}{\hookrightarrow} (q'_c, q'_E) \implies (q'_c, q'_E)\,\mathcal{R}\, q'_a$ with q'_a as above*[10].
3. *$q_c \overset{\tau}{\hookrightarrow}_c q'_c \Longrightarrow (q'_c, q_E)\,\mathcal{R}\, q_a$: states related by τ-transitions refine the same state*

(b) The inverse image under $\overline{\mathcal{R}}$ of any progress condition $P_c = (T_c, T_p)$ of K_{abs}, which is $(\overline{\mathcal{R}}^{-1}(T_c), \overline{\mathcal{R}}^{-1}(T_p))$, is a progress condition of K_{conc}.

Condition (a) ensures that refining an abstract component preserves safety properties. Condition (b) ensures preservation of progress properties.

The last step to obtain a contract framework is to define conformance.

Definition 3.10 (Conformance) *Let $K_\perp = (TS, X, I, Prog)$ be defined as: $TS = (\{q_0\}, q_0, \emptyset, \emptyset)$, $X = \emptyset$, $I_0 = \top$ and $Prog = \emptyset$. We define conformance as refinement in the context of $(K_\perp, \emptyset)$ — i.e., an "empty" with no connectors.*

Theorem 3.11 *We have defined a contract framework. Furthermore, if assumptions are deterministic, then circular reasoning is sound. See Appendix F for a proof.*

4 An application to resource sharing in a network

We apply the proposed methodology to an algorithm for sharing resources in a network presented in [7]. The starting point is both a high-level property and an abstract description of the behavior of an individual node. We represent networks of arbitrary size by a grammar and associating a contract with each node, such that the correctness proof boils down to a set of small verification steps. We consider networks structured as binary trees defining a token ring[11].

Resources shared between nodes are represented by *tokens* circulating in packets containing one or more tokens along the token ring (see figure 3). The *value* of a packet is the number of tokens it contains. A particular token is the *privilege* — denoted P — which allows nodes to accumulate tokens.

[10] If t is independent of the context, i.e., if $P_\gamma = \{p\}$, we use the convention $q_E \overset{\emptyset}{\hookrightarrow}_E q_E$.

[11] We restrict ourselves to binary branching for simplifying the presentation.

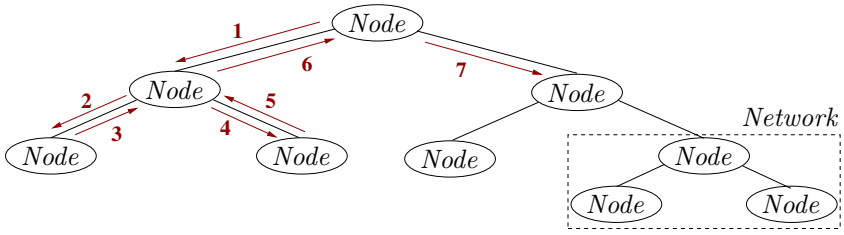


Figure 3: The overall structure of the application.

A node may *request* tokens (Req indicates the numbers of tokens requested). When it has enough tokens for satisfying its request, it is expected to *use* them, and relax the privilege if it has it; when it has resources (tokens) in use, it cannot request additional ones; it may later *free* them or keep them forever. A node can rise a request only when it has no resource in use and no pending request.

Tokens (and the privilege) circulate through ports called get_T (get_P) and $give_T$ ($give_P$), whereas the request, usage or freeing of tokens is indicated through observation ports req, use, $free$. Moreover, a *Node* has state variables indicating whether it has the privilege (P), its number of tokens (Tk), requests (Req) and some port variables used during interactions.

The network is defined by the grammar $\mathcal{G}$, where $\{E_\perp, Node\}$ are terminals and $\{Sys, Net\}$ nonterminals with axiom Sys. The rules are:

$$Sys \longrightarrow \mathcal{I}_{Net}(E_\perp, Net), Net \longrightarrow Node, Net \longrightarrow \mathcal{I}(Node, Net, Net)$$

The connectors of the composition operators $\mathcal{I}$ and $\mathcal{I}_{Net}$ are indicated in Figures 4 and 5. They handle exchange of tokens and privileges and the observation of requested, used, respectively freed tokens.

We assume that connectivity of the network is guaranteed and tokens are never lost. Here, this assumption is encoded in the composition operator. This allows separating completely design and correctness proofs from the resource sharing algorithm and the algorithm guaranteeing connectivity, which is typically implemented in a lower layer of the overall network protocol.

4.0.10 A top-level requirement φ.

We consider here one of the top-level requirements of the algorithm, a progress requirement φ stating that "as long as the requests are reasonable, some of the nodes will be served" — *use* will occur — from time to time. φ is represented in our formalism as depicted in Figure 4, where the second progress property pr_2 says that "it is not possible to switch infinitely often between states S_1 and S_2 (that is, *free* occurs infinitely often) without that a *use* occurs infinitely often as well". "Reasonable" requests means that $0 < R^x \leq Tk$ where R^x is the maximal request and Tk the number of available tokens in the system.

4.0.11 Methodology.

Our goal is to prove that every network built according to grammar $\mathcal{G}$, together with an environment $E_\perp$ giving back tokens and privilege immediately, conforms to φ. For this purpose, we instantiate the methodology of Section 2.0.3:

1. We define $\mathcal{C}_{Node} = (A_{Node}, \mathcal{I}_{Node}, G_{Node})$ and $\mathcal{C}_{Net} = (A_{Net}, \mathcal{I}_{Net}, G_{Net})$ that are contracts for component types *Net* and *Node*.

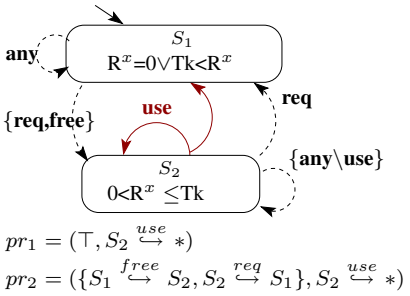
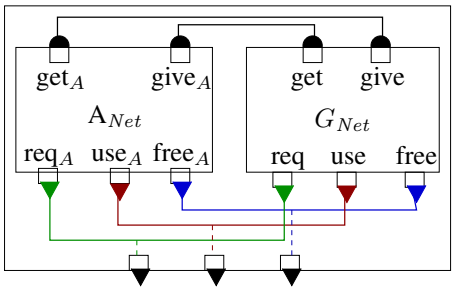


Figure 4: (a) Top-level requirement φ



(b) Composition $\mathcal{I}_{\text{Net}}$ for contract $\mathcal{C}_{\text{Net}}$

2. We show that $\mathcal{I}_{\text{Net}}(A_{\text{Net}}, G_{\text{Net}}) \preccurlyeq \varphi$.

3. We show that $\{\mathcal{C}_{\text{Node}}, \mathcal{C}_{\text{Net}}, \mathcal{C}_{\text{Net}}\}$ dominates $\mathcal{C}_{\text{Net}}$ w.r.t. $\mathcal{I}$.

4. We prove that $E_\perp$ satisfies $\mathcal{C}_{Net}$ and that $Node$ satisfies $\mathcal{C}_{Net}$.

Note that if we want to further refine the *Node* component, we may start by a contract $\mathcal{C}_{Node} = (A_{Net}, \mathcal{I}_{Net}, Node)$. Now, let us give some details.

4.0.12 Interaction models.

Figure 4(b) shows the interaction model $\mathcal{I}_{\text{Net}}$ relating a network — and therefore also a leaf node — to the rest of the system. We represent by get and $give$ respectively port sets $\{get_T, get_P\}$ and $\{give_T, give_P\}$ for token and privilege exchange. $\mathcal{I}$ consists of 3 observation connectors which export a *use* interaction of either the Net or its environment as a global *use* interaction, and analogously for the others. There are also 4 internal connectors for exchanging tokens and privilege. For example, connector $\{give_T[tk] \mid get_{TA}[tk_A], \delta_G : [tk > 0], tk_A := tk\}$ pushes a positive number of tokens from the Network to the environment.

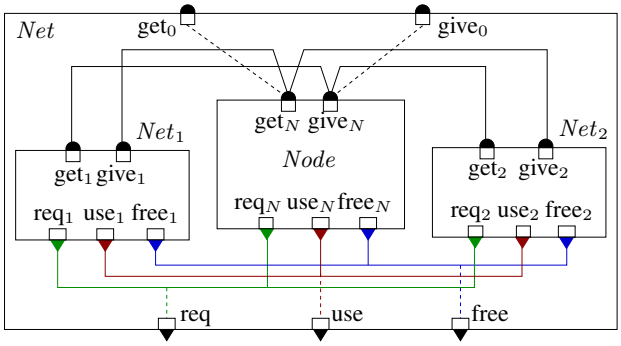
Due to lack of space, we do not present the assumptions and guarantees of the node and network contracts. They are detailed in Appendix F.

Figure 5(a) shows the inner structure of a network component Net. The interaction model $\mathcal{I}$ builds a tree from a (root) node[12] and two networks Net_1, Net_2. Interactions performed by the connectors depicted here are similar to those of Figure 4(b), except that they also ensure that tokens circulate in the correct order.

4.0.13 Experimental results.

To show that $\{\mathcal{C}_{\text{Node}}, \mathcal{C}_{\text{Net}}, \mathcal{C}_{\text{Net}}\}$ dominates $\mathcal{C}_{\text{Net}}$ w.r.t. $\mathcal{I}$, it is sufficient, according to the sufficient condition of section 2.0.3, to prove the conditions given in Figure 5(b). Dominance, conformance and satisfaction problems are reduced to refinement under context checked and discharged automatically by a *Java* tool returning either yes or a trace leading to the violation of refinement.

[12] which is connected in a slightly more complex manner than the leaf node.



1. $\Pi_{\mathcal{P}_{\mathrm{Net}}}(\mathcal{I}(G_{\mathrm{Node}}, G_{\mathrm{Net}}, G_{\mathrm{Net}})) \models \mathcal{C}_{\mathrm{Net}}$

2. $\Pi_{\mathcal{P}_{\mathrm{Node}}}(\mathcal{I}_1(A_{\mathrm{Net}}, G_{\mathrm{Net}}, G_{\mathrm{Net}})) \models \mathcal{C}_{\mathrm{Node}}^{-1}$

3. $\Pi_{\mathcal{P}_{\mathrm{Net}}}(\mathcal{I}_2(A_{\mathrm{Net}}, G_{\mathrm{Node}}, G_{\mathrm{Net}})) \models \mathcal{C}_{\mathrm{Net}}^{-1}$

Figure 5: (a) Inner structure of a network component (b) Sufficient conditions for dominance

5 Discussion and future work

We proposed a design and verification methodology which allows to jointly design and verify safety and progress properties of distributed systems of arbitrary size. We successfully applied it to an algorithm for sharing resources in a tree-shaped network by automatically discharging the required conformance, dominance and satisfaction checks with a prototype tool.

There are several interesting directions to be explored. (a) We have excluded the use of contracts for assume/guarantee reasoning. We may integrate this into the methodology based on the same theory: in our network application, it would be enough to ensure that assumptions express sufficient progress to show conformance of a node contract to "node progress". (b) Extend the methodology to multiple requirements, possibly by using a different decomposition of the system — i.e. a different grammar. (c) Extend the component framework to more general connectors and behaviors to express non functional properties. (d) Build an efficient checker for the different refinement relations, and then, implement tool support for the methodology. We also consider integration into a system design framework — such as SysML promoted by OMG.

References

[1] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Strucutres in Computer Science*, 14(3), 2004. 1

[2] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Proc. of SEFM'06*, pages 3–12. IEEE Computer Society, 2006. 1

[3] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Proc. of FMCO'07*, volume 5382 of *LNCS*, pages 200–225, 2008. 2.0.3

[4] Ph. Bidinger and J.-B. Stefani. The Kell calculus: operational semantics and type systems. In *Proc. of FMOODS*, volume 2884 of *LNCS*, 2003. 1

[5] Simon Bliudze and Joseph Sifakis. The algebra of connectors: structuring interaction in BIP. In *Proc. of EMSOFT'07*, pages 11–20. ACM Press, 2007. 3, 3.0.8, 3.0.8

[6] Marius Bozga, Mohamad Jaber, and Joseph Sifakis. Source-to-source architecture transformation for performance optimization in BIP. In *Proc. of SIES'09*, pages 152–160, 2009. 3, 3.0.8

[7] Ajoy Kumar Datta, Stéphane Devismes, Florian Horn, and Lawrence L. Larmore. Self-stabilizing k-out-of-l exclusion on tree network. *CoRR*, abs/0812.1093, 2008. 4

[8] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proc. of ESEC/SIGSOFT FSE'01*, pages 109–120. ACM Press, 2001. 1, 2.0.3

[9] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Number 54. Cambridge University Press, 2001. 1

[10] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems Vol. 1: Specification*. Springer, 1991. 1

[11] Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992. 1

[12] Sophie Quinton and Susanne Graf. Contract-based verification of hierarchical systems of components. In *Proc. of SEFM'08*, pages 377–381. IEEE Computer Society, 2008. 1, 1.0.1, 2.0.3, B

[13] Joseph Sifakis. A framework for component-based construction. In *Proc. of SEFM'05*, pages 293–300. IEEE Computer Society, 2005. 2.0.2, 2.0.3, 2.0.3, 3

[14] Z. Stadler and O. Grumberg. Network grammars, communication behaviours and automatic verification. In *Proc. of AVMFSS*, volume 407 of *LNCS*, 1989. 2.0.4

Appendix

We expect the paper to be self-contained, and we present in these appendix some material that should allow to assess the correctness of the framework and the methodology, and to give a bit more insight into the application. Shortly, we will make available a complete version of the paper including this material.

This appendix contains:

1. properties expected from a well-defined projection
2. formal definitions of compositionality and soundness of circular reasoning
3. a proof of theorem 2.6
4. rules for inferring new progress properties from existing ones
5. definitions of merge of connectors and composition of interaction models.
6. a proof of theorem 3.11
7. assumptions and guarantees for components Node and Net.

A Properties expected from a well-defined projection

Definition A.1 (Projection) Π *is a* projection *iff for any components K_i and A on disjoint interfaces $\mathcal{P}$ and $\mathcal{P}_A$, and any composition operator gl on $\mathcal{P} \cup \mathcal{P}_A$:*

1. *for $P' \subseteq \mathcal{P}$ and any gl_1, gl_2 with $P_{gl_1} = P' \cup \mathcal{P}_A$, $P_{gl_2} = \mathcal{P} \backslash P'$ s.t. $gl = gl_1 \circ gl_2$:*

$$K_1 \sqsubseteq_{A,gl} K_2 \wedge \Pi_{P'}(K_2) \sqsubseteq_{A,gl_1} G \implies \Pi_{P'}(K_1) \sqsubseteq_{A,gl_1} G$$

Note that G is defined on P'.

2. *for $P'_A \subseteq \mathcal{P}_A$ and any gl_1, gl_2 on $P_{gl_1} = \mathcal{P} \cup P'_A$, $P_{gl_2} = \mathcal{P}_A \backslash P'_A$ s.t. $gl = gl_1 \circ gl_2$:*

$$K \sqsubseteq_{\Pi_{P'_A}(A),gl_1} G \implies K \sqsubseteq_{A,gl} G$$

Note that G is defined on P.

These properties state that ports of the component (and symmetrically of the environment) which do not appear in interactions with the environment (resp. the component) may be abstracted away when checking refinement under context. Note that we might use an equivalence relation between composition operators rather than equality. This would be, e.g., in the context of BIP connectors, an equivalence based on equality of the associated sets of interactions.

B Compositionality and circular reasoning

Definition B.1 (Compositionality) *Refinement under context $\{\sqsubseteq_{E,gl}\}$ is* compositional w.r.t. composition *iff for any context (E, gl) for an interface $\mathcal{P}$ and gl_E, E_1, E_2 such that $E = gl_E(E_1, E_2)$, the following holds for any K_1, K_2 on $\mathcal{P}$:*

$$K_1 \sqsubseteq_{gl_E(E_1,E_2),gl} K_2 \implies gl_1(K_1, E_1) \sqsubseteq_{E_2,gl_2} gl_1(K_2, E_1)$$

where gl_1 and gl_2 are obtained from gl and gl_E using flattening and structuring.

Definition B.2 (Sound circular reasoning) Circular reasoning *is sound for a refinement under context $\{\sqsubseteq_{E,gl}\}$ iff it is such that, given an interface $\mathcal{P}$, a component K on $\mathcal{P}$, a context (E, gl) and a contract $\mathcal{C} = (A, gl, G)$ for $\mathcal{P}$, we have:*

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

For a particular framework, this property can be proved by an induction based on the semantics of composition and refinement. For example, in the framework based on I/O automata defined in [12], circular reasoning is sound because exactly one behavior has control over each interaction.

C Sufficient condition for dominance (Theorem 2.6)

Theorem 2.6 *If circular reasoning is sound and* $\forall i.\ \exists gl_{E_i}.\ gl \circ gl_I = gl_i \circ gl_{E_i}$*, then to prove that* $\mathcal{C}$ *dominates* $\{\mathcal{C}_i\}_{i=1..n}$ *w.r.t. gl, it is sufficient to prove that:*

$$\begin{cases} \Pi_P(gl_I\{G_1, \dots, G_n\}) \models \mathcal{C} \\ \forall i, \Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, G_1, \dots, G_{i-1}, G_{i+1}, \dots, G_n\}) \models \mathcal{C}_i^{-1} \end{cases}$$

Proof For all $i = 1..n$, let K_i be a component on $\mathcal{P}_i$. Suppose the following:

1. $\forall i.\ \exists gl_{E_i}.\ gl \circ gl_I = gl_i \circ gl_{E_i}$
2. $\Pi_P(gl_I\{G_1, \dots, G_n\}) \sqsubseteq_{A,gl} G$
3. $\forall i, \Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, G_1, \dots, G_{i-1}, G_{i+1}, \dots, G_n\}) \sqsubseteq_{G_i,gl_i} A_i$
4. $\forall i, K_i \sqsubseteq_{A_i,gl_i} G_i$

We aim at proving $\Pi_P(gl_I\{K_1, \dots, K_n\}) \models \mathcal{C}$, i.e., $\Pi_P(gl_I\{K_1, \dots, K_n\}) \sqsubseteq_{A,gl} G$. For this, we show by induction that for all l in $[\![0, n]\!]$, for all partition $\{J, K\}$ of $[\![1, n]\!]$ such that $|J| = l$:

$$\begin{cases} \Pi_P(gl_I\{\mathcal{K}_J \cup \mathcal{G}_K\}) \sqsubseteq_{A,gl} G \\ \forall i \in K, \Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, \mathcal{E}^i_{J,K}\}) \sqsubseteq_{G_i,gl_i} A_i \end{cases}$$

with $\mathcal{K}_J = \{K_j\}_{j\in J}$, $\mathcal{G}_K = \{G_k\}_{k\in K}$ and $\mathcal{E}^i_{J,K} = \mathcal{K}_J \cup (\mathcal{G}_K \backslash \{G_i\})$.

- $l = 0$. By (2) and (3) the property holds.
- $0 \leq l < n$. We suppose that our property holds for l. Let $\{J', K'\}$ be a partition of $[\![1, n]\!]$ such that $|J'| = l + 1$. Let q be an element of J'. We fix $J = J' \backslash \{q\}$ and $K = K' \cup \{q\}$.

Step 1. We first prove that $\Pi_P(gl_I\{\mathcal{K}_{J'} \cup \mathcal{G}_{K'}\}) \sqsubseteq_{A,gl} G$.

$$\begin{cases} K_q \sqsubseteq_{A_q,gl_q} G_q \text{ from (5)} \\ \Pi_{\mathcal{P}_{A_q}}(gl_{E_q}\{A, \mathcal{E}^q_{J,K}\}) \sqsubseteq_{G_q,gl_q} A_q \text{ (recurrence hypothesis, as } q \in K) \end{cases}$$

Hence, by applying circular reasoning: $K_q \sqsubseteq_{\Pi_{\mathcal{P}_{A_q}}(gl_{E_q}\{A,\mathcal{E}^q_{J,K}\}),gl_q} G_q$.
By (1) and property 2. of Definition A.1, this implies: $K_q \sqsubseteq_{gl_{E_q}\{A,\mathcal{E}^q_{J,K}\},gl\circ gl_I} G_q$.
As refinement is compositional, we get: $gl_I\{K_q, \mathcal{E}^q_{J,K}\} \sqsubseteq_{A,gl} gl_I\{G_q, \mathcal{E}^q_{J,K}\}$.
This is equivalent to $gl_I\{\mathcal{K}_{J'} \cup \mathcal{G}_{K'}\} \sqsubseteq_{A,gl} gl_I\{\mathcal{K}_J \cup \mathcal{G}_K\}$.
Finally, by using property 1. of Definition A.1 and the recurrence hypothesis, we obtain: $\Pi_P(gl_I\{\mathcal{K}_{J'} \cup \mathcal{G}_{K'}\}) \sqsubseteq_{A,gl} G$.

Step 2. We now prove that $\forall i \in K'.\ \Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, \mathcal{E}^i_{J',K'}\}) \sqsubseteq_{G_i,gl_i} A_i$.
We fix $i \in K'$. We have proved in step 1 that $K_q \sqsubseteq_{gl_{E_q}\{A,\mathcal{E}^q_{J,K}\},gl\circ gl_I} G_q$.
$K = K' \cup \{q\}$ so $i \in K$. Thus, by compositionality of refinement under context: $gl_{E_i}\{K_q, A, \mathcal{E}^q_{J,K\backslash\{i\}}\} \sqsubseteq_{G_i,gl_i} gl_{E_i}\{G_q, A, \mathcal{E}^q_{J,K\backslash\{i\}}\}$.
This boils down to $gl_{E_i}\{A, \mathcal{E}^i_{J',K'}\} \sqsubseteq_{G_i,gl_i} gl_{E_i}\{A, \mathcal{E}^i_{J,K}\}$.
Hence, using property 1. of projection and the recurrence hypothesis: $\Pi_{\mathcal{P}_{A_i}}(gl_{E_i}\{A, \mathcal{E}^i_{J',K'}\}) \sqsubseteq_{G_i,gl_i} A_i$.

Conclusion By applying our property to $l = n$: $\Pi_P(gl_I\{K_1, \dots, K_n\}) \sqsubseteq_{A,gl} G$. □

D Inference of progress properties

The following are monotonicity properties which allow inferring new progress properties from existing ones.

1. If (T_c, T_p) and (T_c, T'_p) are progress properties for K, then so is $(T_c, T_p \cup T'_p)$; that is, the smaller the set T_p, the stronger the progress constraint.
2. The opposite implication can only be guaranteed when either T_1 or T_2 is never enabled in K.
3. If $(T_c \cup T'_c, T_p)$ is a progress property for K, then so is (T_c, T_p); that is, the larger the set T_c, the stronger the progress constraint.
4. The opposite implication is only guaranteed if K has no loop in T'_c or if (T'_c, T_p) also a progress transition.
5. If $(\top, T_p \cup T'_p)$ is a progress property and $start(T_p) \cap start(T'_p) = \emptyset$, then $(\top, T_p)$ and $(\top, T'_p)$ are also progress properties.
6. If $t_1 = q \xrightarrow{\tau} q'$, $t_2 = q' \xrightarrow{p} q''$ are transitions and $(\top, \{t_1\})$, $(\top, \{t_2\})$ are unconditional progress properties, then so is $(\top, \{q \xrightarrow{p} q''\})$.

E Composition of interaction models

Definition E.1 (Merge of rendez-vous connectors) *Let γ_1 and γ_2 be two rendez-vous connectors such that $P_1 \cap P_2 = \emptyset$ and $p_1 \neq p_2$. The merge of γ_1 and γ_2, denoted $\gamma_1 \bullet \gamma_2$, is defined in the following situations:*

- *if $(p_1 \notin P_2$ and $p_2 \in P_1)$ then $\gamma_1 \bullet \gamma_2 = (p[x], P, \delta)$ with:*
  - $p = p_1$
  - $P = P_1 \cup P_2 \setminus \{p_2\}$
  - $\delta = (G, \mathcal{U}, \mathcal{D})$ *is defined as follows:*
    - $G = G_2 \wedge \exists v_2.\ G_1[v_2/x_2] \wedge \mathcal{U}_2[v_2/x_2]$
    - $\mathcal{U} = \exists v_2.\ \mathcal{U}_2[v_2/x_2] \wedge \mathcal{U}_1[v_2/x_2]$
    - $\mathcal{D}_{x_k} = \begin{cases} \mathcal{D}_{1,x_k} & \text{if } x_k \in P_1 \setminus \{x_2\} \\ \exists v_2.\ \mathcal{D}_{1,x_2}[v_2/x_2] \wedge \mathcal{D}_{2,x_k}[v_2/x_2] & \text{if } x_k \in P_2 \end{cases}$
- *if $(p_1 \in P_2$ and $p_2 \notin P_1)$ then $\gamma_1 \bullet \gamma_2 = \gamma_2 \bullet \gamma_1$.*

Definition E.2 (Merge of observation connectors) *Let γ_1 and γ_2 be two observation connectors such that $P_1 \cap P_2 = \emptyset$ and $p_1 \neq p_2$. The merge of γ_1 and γ_2, denoted $\gamma_1 \bullet \gamma_2$, is defined in the following situations:*

- *if $(p_1 \notin P_2$ and $p_2 \in P_1)$ then $\gamma_1 \bullet \gamma_2 = (p, P, \delta)$ with:*
  - $p = p_1$
  - $P = P_1 \cup P_2 \setminus \{p_2\}$
- *if $(p_1 \in P_2$ and $p_2 \notin P_1)$ then $\gamma_1 \bullet \gamma_2 = \gamma_2 \bullet \gamma_1$.*

If $(p_1 \in P_2$ and $p_2 \in P_1)$, then $\gamma_1 \bullet \gamma_2$ is not defined because this would result in a cyclic connector. Besides, if $(p_1 \notin P_2$ and $p_2 \notin P)$, then γ_1 and γ_2 are disjoint, thus they cannot be merged. Note that merge of connectors is by definition commutative and it is applied only on connectors with the same type. This definition extends naturally to interaction models in general.

Definition E.3 (Composition of interaction models) *Let $\mathcal{I}$, $\mathcal{I}'$ be two interaction models defined respectively on P and P', where $\mathcal{I} = \{\gamma\}$ and $\mathcal{I}' = \{\gamma'_1,\ \gamma'_2\}$. The composition of $\mathcal{I}$ and $\mathcal{I}'$, denoted $\mathcal{I} \circ \mathcal{I}'$, is an interaction model defined on $P \cup P' \setminus \{p, p'_1, p'_2\}$ as follows:*

- *if $p \notin (P'_1 \cup P'_2)$*
  - *if $\{p'_1,\ p'_2\} \cap P = \emptyset$ then $\mathcal{I} \circ \mathcal{I}' = \{\gamma,\ \gamma'_1,\ \gamma'_2\}$;*
  - *if $\{p'_1,\ p'_2\} \cap P = \{p'_1\}$ then $\mathcal{I} \circ \mathcal{I}' = \{\gamma \bullet \gamma'_1,\ \gamma'_2\}$;*
  - *if $\{p'_1,\ p'_2\} \cap P = \{p'_2\}$ then $\mathcal{I} \circ \mathcal{I}' = \{\gamma'_1,\ \gamma \bullet \gamma'_2\}$;*
  - *if $\{p'_1,\ p'_2\} \cap P = \{p'_1,\ p'_2\}$ then $\mathcal{I} \circ \mathcal{I}' = \{(\gamma \bullet \gamma'_1) \bullet \gamma'_2\}$;*
- *if $p \in P'_1$ and $p \notin P'_2$*
  - *if $\{p'_1,\ p'_2\} \cap P = \emptyset$ then $\mathcal{I} \circ \mathcal{I}' = \{\gamma \bullet \gamma'_1,\ \gamma'_2\}$;*
  - *if $\{p'_1,\ p'_2\} \cap P = \{p'_2\}$ then $\mathcal{I} \circ \mathcal{I}' = \{(\gamma \bullet \gamma'_1) \bullet \gamma'_2\}$;*

- ⋆ *if* $p \notin P_1'$ *and* $p \in P_2'$
  - ⋆ *if* $\{p_1',\ p_2'\} \cap P = \emptyset$ *then* $\mathcal{I} \circ \mathcal{I}' = \{\gamma_1',\ \gamma \bullet \gamma_2'\}$;
  - ⋆ *if* $\{p_1',\ p_2'\} \cap P = \{p_1'\}$ *then* $\mathcal{I} \circ \mathcal{I}' = \{(\gamma \bullet \gamma_1') \bullet \gamma_2'\}$.
- *Otherwise,* $\mathcal{I} \circ \mathcal{I}'$ *is undefined as it induces cyclic connectors.*

Conditions introduced by the symbol $\star$ are redundant since they can be obtained by exchanging γ_1 and γ_2 in the definition.

F Well-definedness of the contract framework (Theorem 3.11)

We detail here the proof steps required to show that we have indeed a verification framework in which circular reasoning is sound. We do not detail all the proofs here, but rather give intuitions. The proof of circular reasoning, which is the most complicated one, is fully explained.

Component algebra.

- $(GL, \circ)$ is a commutative monoid, i.e. it ensures associativity, identity and of course commutativity:
  1. $\forall \mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3 \in GL.\ (\mathcal{I}_1 \circ \mathcal{I}_2) \circ \mathcal{I}_3 = \mathcal{I}_1 \circ (\mathcal{I}_2 \circ \mathcal{I}_3)$
  2. $\exists \mathcal{I}_e \in GL.\ \forall \mathcal{I} \in GL.\ \mathcal{I}_e \circ \mathcal{I} = \mathcal{I} \circ \mathcal{I}_e = \mathcal{I}$
  3. $\forall \mathcal{I}_1, \mathcal{I}_2 \in GL.\ \mathcal{I}_1 \circ \mathcal{I}_2 = \mathcal{I}_2 \circ \mathcal{I}_1$
- Definitions of interfaces are consistent with those of Definition 2.1.
- Flattening and structuring are always possible:
  1. $\mathcal{I}\{\mathcal{I}_1\{\mathcal{K}_1\}, \mathcal{I}_2\{\mathcal{K}_2\}\} \cong (\mathcal{I} \circ \mathcal{I}_1 \circ \mathcal{I}_2)\{\mathcal{K}_1 \cup \mathcal{K}_2\}$
     In other words, composition of behaviors is consistent with composition of composition operators.
  2. $\forall \mathcal{I}.\ \exists \mathcal{I}', \mathcal{I}_1, \mathcal{I}_2.\ \mathcal{I}\{\mathcal{K}_1 \cup \mathcal{K}_2\} \cong \mathcal{I}'\{\mathcal{I}_1\{\mathcal{K}_1\}, \mathcal{I}_2\{\mathcal{K}_2\}\}$
     A composition operator $\mathcal{I}$ can always be decomposed into $\mathcal{I}' \circ \mathcal{I}_1 \circ \mathcal{I}_2$.

As we use BIP connectors, all these properties are satisfied. We only needed to require associativity of up and down predicates of rendez-vous connectors.

Contract framework.

- Compositionality: $K_1 \sqsubseteq_{\mathcal{I}_E\{E_1,E_2\},\mathcal{I}} K_2 \implies \mathcal{I}_1\{K_1, E_1\} \sqsubseteq_{E_2,\mathcal{I}_2} \mathcal{I}_1\{K_2, E_1\}$ where $\mathcal{I}_1$ and $\mathcal{I}_2$ are the composition operators obtained from $\mathcal{I}$ and $\mathcal{I}_E$ using flattening and structuring.
- Conformance is consistent with refinement under context:
  $K_1 \sqsubseteq_{E,\mathcal{I}} K_2 \implies \mathcal{I}\{K_1, E\} \preccurlyeq \mathcal{I}\{K_2, E\}$. This is trivial given compositionality.
- Projection is consistent with Definition A.1. This is a classical result.

Circular reasoning.

- Circular reasoning is sound: $K \sqsubseteq_{A,\mathcal{I}} G \wedge E \sqsubseteq_{G,\mathcal{I}} A \implies K \sqsubseteq_{E,\mathcal{I}} G$.

Proof Let K be a component on an interface P, $(E, \mathcal{I})$ a context for P and $\mathcal{C} = (A, \mathcal{I}, G)$ a contract for P. We suppose that $K \sqsubseteq_{A,\mathcal{I}} G \wedge E \sqsubseteq_{G,\mathcal{I}} A$. We have to prove that $K \sqsubseteq_{E,\mathcal{I}} G$.

Given two relations $\alpha_1(X_K, X_G)$ and $\alpha_2(X_E, X_A)$ and as $K \sqsubseteq_{A,\mathcal{I}} G$ and $E \sqsubseteq_{G,\mathcal{I}} A$, there exist two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ on respectively $(Q_K \times Q_A) \times Q_G$ and $(Q_E \times Q_G) \times Q_A$ verifying the conditions of Definition 3.9.

We define $\mathcal{R} \subseteq (Q_K \times Q_E) \times Q_G$ as follows: for $q_K \in Q_K$, $q_E \in Q_E$, $q_G \in Q_G$, we define $(q_K, q_E)\, \mathcal{R}\, q_G$ iff there exists q_A such that $(q_K, q_A)\, \mathcal{R}_1\, q_G$ and $(q_E, q_G)\, \mathcal{R}_2\, q_A$. ($\mathcal{V}_i$ denotes a valuation of X_i)

Safety part. We have $(q_K^0, q_A^0)\,\mathcal{R}_1\,q_G^0$ and for all $q_K \in Q_K, q_G \in Q_G, q_A \in Q_A$, $(q_K, q_A)\,\mathcal{R}_1\,q_G$ implies:

1. $I_{q_K}(X_K) \wedge \alpha^1(X_K, X_G) \implies I_{q_G}(X_G)$
2. $\forall p[x] \in P$, the following holds :
   (a) for any value v of x: $\exists t_K = q_K \xrightarrow{p}_K q'_K$ and $\mathcal{V}_K, \mathcal{V}_K^{new}$ satisfying Sem_{t_K} implies $\exists q'_G, t_G = q_G \xrightarrow{p}_G q'_G$ and $\mathcal{V}_G, \mathcal{V}_G^{new}$ consistent with α^1 and satisfying Sem_{t_G}.
   (b) $\exists \gamma.\ P_\gamma = \{p, p_A\} \wedge (q_K, q_A) \overset{p_\gamma}{\hookrightarrow} (q'_K, q'_A) \implies (q'_K, q'_A)\,\mathcal{R}_1\,q'_G$ with q'_G as above.
3. $q_K \overset{\tau}{\hookrightarrow}_K q'_K \implies (q'_K, q_A)\,\mathcal{R}_1\,q_G$

Besides, we have $(q_E^0, q_G^0)\,\mathcal{R}_2\,q_A^0$ and for all $q_E \in Q_E, q_A \in Q_A, q_G \in Q_G$, $(q_E, q_G)\,\mathcal{R}_2\,q_A$ implies:

1. $I_{q_E}(X_E) \wedge \alpha^2(X_E, X_A) \implies I_{q_A}(X_A)$
2. $\forall p[x] \in P$, the following holds :
   (a) for any value v of x: $\exists t_E = q_E \xrightarrow{p}_E q'_E$ and $\mathcal{V}_E, \mathcal{V}_E^{new}$ satisfying Sem_{t_E} implies $\exists q'_A, t_A = q_A \xrightarrow{p}_A q'_A$ and $\mathcal{V}_A, \mathcal{V}_A^{new}$ consistent with α^2 and satisfying Sem_{t_A}.
   (b) $\exists \gamma.\ P_\gamma = \{p, p_G\} \wedge (q_E, q_G) \overset{p_\gamma}{\hookrightarrow} (q'_E, q'_G) \implies (q'_E, q'_G)\,\mathcal{R}_2\,q'_A$ with q'_A as above.
3. $q_E \overset{\tau}{\hookrightarrow}_E q'_E \implies (q'_E, q_G)\,\mathcal{R}_2\,q_A$

We prove now that $\mathcal{R}$ is the relation we are looking for. Let $q_K \in Q_K$, $q_E \in Q_E$, $q_G \in Q_G$ be such that $(q_K, q_E)\,\mathcal{R}\,q_G$. Let q_A be such that $(q_K, q_A)\,\mathcal{R}_1\,q_G$ and $(q_E, q_G)\,\mathcal{R}_2\,q_A$. We have to prove that:

1. $I_{q_K}(X_K) \wedge \alpha'(X_K, X_G) \implies I_{q_G}(X_G)$
2. $\forall p[x] \in P$, the following holds :
   (a) for any value v of x: $\exists t_K = q_K \xrightarrow{p}_K q'_K$ and $\mathcal{V}_K, \mathcal{V}_K^{new}$ satisfying Sem_{t_K} implies $\exists q'_G, t_G = q_G \xrightarrow{p}_G q'_G$ and $\mathcal{V}_G, \mathcal{V}_G^{new}$ consistent with α and satisfying Sem_{t_G}.
   (b) $\exists \gamma.\ P_\gamma = \{p, p_E\} \wedge (q_K, q_E) \overset{p_\gamma}{\hookrightarrow} (q'_K, q'_E) \implies (q'_K, q'_E)\,\mathcal{R}\,q'_G$ with q'_G as above.
3. $q_K \overset{\tau}{\hookrightarrow}_K q'_K \implies (q'_K, q_E)\,\mathcal{R}\,q_G$

- Condition 1. Given $\alpha' = \alpha^1$, condition 1 is the same as condition 1 of $\mathcal{R}_1$
- Condition 2.
  Part (a): Deducted from condition 2, part (a) for $\mathcal{R}_1$.
  Part (b): Let us suppose that $q_K \overset{p}{\hookrightarrow}_K q'_K \wedge\ q_E \overset{p_E}{\hookrightarrow}_E q'_E \wedge\ (\exists \gamma\ s.t.\ P_\gamma = \{p,\ p_E\})$. Condition 2 for $\mathcal{R}_2$ implies that $\exists q'_A \in Q_A\ s.t.\ q_A \overset{p_E}{\hookrightarrow}_A q'_A$.
  Hence: $q_K \overset{p}{\hookrightarrow}_K q'_K\ \wedge\ q_A \overset{p_E}{\hookrightarrow}_A q'_A \wedge\ (\exists \gamma\ s.t.\ P_\gamma = \{p,\ p_E\})$.
  Thus Condition 2 for $\mathcal{R}_1$ implies that there exists a q''_G s.t. $(q_G \overset{p}{\hookrightarrow}_G q''_G)$ and $\forall q'_A, q_A \overset{p_E}{\hookrightarrow}_A q'_A \implies (q'_K, q'_A)\,\mathcal{R}_1\,q''_G$. We now want to prove that $\forall q'_E,\ q_E \overset{p_E}{\hookrightarrow}_E q'_E \implies (q'_K, q'_E)\,\mathcal{R}\,q''_G$. Let us fix q'_E such that $q_E \overset{p_E}{\hookrightarrow}_E q'_E$. So we have :

- $q_E \overset{p_E}{\hookrightarrow}_E q'_E$
- $(\exists \gamma \; s.t. \; P_\gamma = \{p, \; p_E\})$ and $q_K \overset{p}{\hookrightarrow}_K q'_K$,

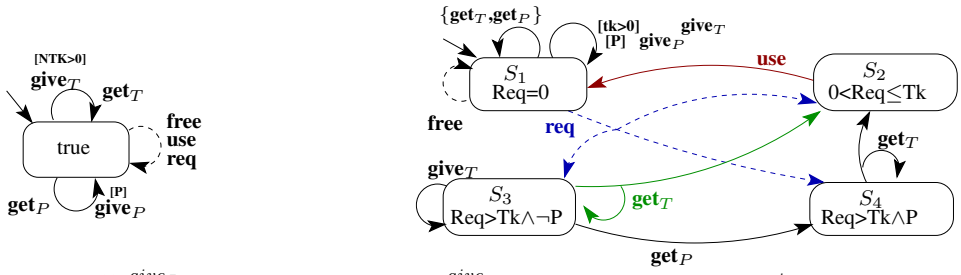
Thus condition 2 part (2) for $\mathcal{R}_1$ implies that $\exists q''_G \in Q_G \; s.t. \; q_G \overset{p}{\hookrightarrow}_G q''_G$ So condition 2 for $\mathcal{R}_2$ implies that it exists a q''_A such that $q_A \overset{p}{\hookrightarrow}_A q''_A$ and $\forall q'_G, \; q_G \overset{p}{\hookrightarrow}_G q'_G \implies (q'_E, q'_G) \, \mathcal{R}_2 \, q''_A$. We apply this relation to q''_G and Similarly we apply $\forall q'_A, \; q_A \overset{p_E}{\hookrightarrow}_A q'_A \implies (q'_K, q'_A) \, \mathcal{R}_1 \, q''_G$ to q''_A, Thus we get:

- $(q'_E, q''_G) \, \mathcal{R}_2 \, q''_A$
- $(q'_K, q''_A) \, \mathcal{R}_1 \, q''_G$

Hence $(q'_K, q'_E) \, \mathcal{R} \, q''_G$.

- Condition 3. Let us suppose that $q_K \overset{\tau}{\hookrightarrow}_K q'_K$. Thus condition 3 of $\mathcal{R}_1$ implies that $(q'_K, q_A) \, \mathcal{R}_1 \, q_G$. Besides we have $(q_E, q_G) \, \mathcal{R}_2 \, q_A$. Hence $(q'_K, q_E) \, \mathcal{R} \, q_G$

Progress part. As the progress part does not involve the context, the progress part for $\mathcal{R}$ is deduced from the progress properties for $\mathcal{R}_1$. □

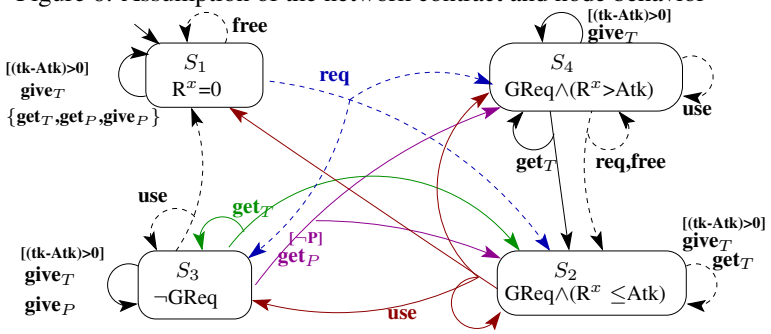


$pr_1^A = (\overset{use}{\hookrightarrow}, \overset{give_P}{\hookrightarrow})$
$pr_2^A = (\top, \{\overset{use}{\hookrightarrow}, \overset{get_T}{\hookrightarrow}, \overset{get_P}{\hookrightarrow}\})$
$pr_3^A = (\top, [\neg P] \overset{give_T}{\hookrightarrow})$

$pr_1^N = (\top, S_i \overset{give_T}{\hookrightarrow} *)_{i\in\{1,3\}}$
$pr_2^N = (\top, S_i \overset{get_T}{\hookrightarrow} *)_{i\in\{1,3,4\}}$
$pr_3^N = (\top, S_i \overset{get_P}{\hookrightarrow} *)_{i\in\{1,3\}}$

$pr_4^N = (\top, S_1 \overset{give_P}{\hookrightarrow} S_1)$
$pr_5^N = (\top, S_2 \overset{use}{\hookrightarrow} S_1)$
$pr_6^N = (S_4 \overset{get_T}{\hookrightarrow} S_4, S_4 \overset{get_T}{\hookrightarrow} S_2)$

Figure 6: Assumption of the network contract and node behavior



$(\top, S_i \overset{give_T}{\hookrightarrow} *)_{i=\{1,3\}}$
$(\top, S_i \overset{get_T}{\hookrightarrow} *)_{i\in\{1,3,4\}}$
$(\top, S_i \overset{get_P}{\hookrightarrow})_{i\in\{1,3\}}$

$(\top, S_i \overset{give_P}{\hookrightarrow} *)_{i\in\{1,3\}}$
$(\top, S_2 \overset{use}{\hookrightarrow} *)$
$(S_4 \overset{get_T}{\hookrightarrow} S_4, S_4 \overset{get_T}{\hookrightarrow} S_2)$

$(\{S_3 \overset{get_T}{\hookrightarrow} S_3\} \setminus \{S_3 \overset{give_T}{\hookrightarrow} S_3\}, S_3 \overset{use}{\hookrightarrow} S_3)$
$(\{S_i \overset{use}{\hookrightarrow} *\}_{i\in\{2,4\}}, \{S_i \overset{give_P}{\hookrightarrow} *\}_{i\in\{1,3\}})$

Figure 7: Guarantee of the network contract

G Assumptions and guarantees for Node and Net

The contracts for the Net and Node components are depicted in Figures 6 and 7. They define the corresponding local transitions $give_T$ and get_{TA} which decrease, respectively increase, the local state variable Tk of the network and the environment; transition $give_T$ has a guard to make sure that Tk is never negative.

Assumption A_{Net} represents the environment of an arbitrary network component. It cannot keep tokens or P indefinitely if it does infinitely many *use* .

Note that the guarantee of the network contract is slightly more nondeterministic and more complex than the node guarantee. In particular, it is not enough that a network has a privilege for being able to collect tokens. The ability of a network to collect tokens is indicated by $GReq$, a state variable of the hierarchical network component which — for the dominance proof — is defined in terms of the state variables of the inner structure of a network (see Figure 5).