Interface Theories with Component Reuse

Laurent Doyen
EPFL, Switzerland
laurent.doyen@epfl.ch

Thomas A. Henzinger
EPFL, Switzerland
tah@epfl.ch

Barbara Jobstmann
EPFL, Switzerland
barbara.jobstmann@epfl.ch

Tatjana Petrov
EPFL, Switzerland
tatjana.petrov@epfl.ch

ABSTRACT

Interface theories have been proposed to support incremental design and independent implementability. Incremental design means that the compatibility checking of interfaces can proceed for partial system descriptions, without knowing the interfaces of all components. Independent implementability means that compatible interfaces can be refined separately, maintaining compatibility. We show that these interface theories provide no formal support for component reuse, meaning that the same component cannot be used to implement several different interfaces in a design. We add a new operation to interface theories in order to support such reuse. For example, different interfaces for the same component may refer to different aspects such as functionality, timing, and power consumption. We give both stateless and stateful examples for interface theories with component reuse. To illustrate component reuse in interface-based design, we show how the stateful theory provides a natural framework for specifying and refining PCI bus clients.

Categories and Subject Descriptors

D.2.12 [Software Engineering]: Interoperability—Interface definition languages

General Terms

Design, Theory

Keywords

Interfaces, Composition, Refinement

1. INTRODUCTION

Interface theories [5] are intended to provide a formal framework for component-based design. An interface constrains the interaction of a component with its environment.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT’08, October 19–24, 2008, Atlanta, Georgia, USA.
Copyright 2008 ACM 978-1-60558-468-3/08/10 ...$5.00.
entire design $I_{111} || I_{112} || I_{113} || I_{21} || I_{22}$ are guaranteed to fit together. Design, however, rarely proceeds in such a strict, tree-like, top-down manner. Often design involves the use of already available components. Also, for space or cost reasons, different logical parts of a design may have to share a common implementation. In Figure 1, suppose the interfaces $I_{22}$ and $I_{112}$ (which are at different levels in different parts of the design tree) are similar enough that they should be implemented by the same component. Interface theories do not provide for this possibility. In this paper, we extend interface theories to support such shared implementability of interfaces. In other words, within our extended theories, components can be reused in different parts of a design (or in different designs); designs are not restricted to be trees of components, but they can be DAGs.

Formally, we say that two interfaces $I_1 = (\phi_1, \psi_1)$ and $I_2 = (\phi_2, \psi_2)$ are shared refinable if there is an interface that refines both $I_1$ and $I_2$. If $I_1$ and $I_2$ are shared refinable, then we compute the shared refinement $I_1 \cap I_2$ as the most general refinement of $I_1$ and $I_2$, i.e., the greatest lower bound in the refinement lattice on interfaces: the assumption of $I_1 \cap I_2$ is $\phi_1 \lor \phi_2$, and the guarantee of $I_1 \cap I_2$ is $\psi_1 \land \psi_2$. A component can be used to implement both $I_1$ and $I_2$ iff it refines $I_1 \cap I_2$. Notice that such a component must be prepared to accept inputs that satisfy any of the two assumptions $\phi_1$ and $\phi_2$, and it must provide outputs that satisfy both guarantees $\psi_1$ and $\psi_2$. Interestingly, the shared refinement $I_1 \cap I_2$ can also be used to implement two or more different views (or aspects) of a single component. For example, the interface $I_1$ may provide functional constraints (i.e., assumptions and guarantees) and the interface $I_2$ may provide power constraints on the same component. For the component to satisfy both the constraints specified by $I_1$ and the constraints specified by $I_2$, it must refine $I_1 \cap I_2$. Note that this is different from refining the composition $I_1 || I_2$: while the composition $I_1 || I_2$ has the assumption $\phi_1 \land \phi_2$, the shared refinement has the assumption $\phi_1 \lor \phi_2$.

In Section 2, we formally develop a simple, stateless interface theory with shared refinement along the lines outlined above. The formalism becomes far more interesting and powerful once we introduce a temporal aspect in the environment strategy of the more abstract automaton. In Section 3, we add shared refinement to such a stateful theory of interfaces. For technical simplicity, we choose to present shared refinement not for the original asynchronous theory of interface automata [4], but for a synchronous version, which was used in [2] to specify interfaces for PCI bus components.

As is to be expected from the stateless case, shared refinement is different from parallel composition also in the automaton case. The standard parallel composition of two automata with $n_1$ and $n_2$ states is the product automaton with $n_1n_2$ states. The parallel composition of two interface automata is a pruned product automaton, from which states without a winning environment strategy are removed. By contrast, the shared refinement of two interface automata with $n_1$ and $n_2$ states is an extended product automaton with up to $n_1n_2 + n_1 + n_2$ states. Roughly speaking, as long as the assumptions of both automata are satisfied, the shared refinement provides the guarantees of both automata. But as soon as the assumption of one automaton is violated, only the guarantee of the other automaton is maintained. We believe that this automata-theoretic construction is interesting in its own right, for example, to combine two different views of a system. We illustrate its use in Section 4, where we provide a refined version of the PCI bus example from [2]. In particular, we provide a functional interface $F_1$ and a power interface $P$ for a PCI bus client, as well as a functional interface $F_2$ for the client of a different bus. Then we show that $F_1 \cap P$ and $F_2 \cap P$ are shared refinable. In other words, the client specifications for both busses can be implemented by the same component, and the requirement on such a component is that it refines the shared refinement $F_1 \cap F_2 \cap P$.

2. STATELESS INTERFACES

A stateless interface for a component in a design describes the environments in which the component can be embedded [4, 8]. It has input and output variables, and two predicates restricting their values: the first predicate specifies the set of input values that the component should accept, the second predicate specifies the set of output values that the component may produce.

Definition 1 (Stateless Interface). A stateless interface is a tuple $M = \langle X^I, X^O, \varphi, \psi \rangle$, where

- $X^I$ and $X^O$ are two disjoint sets of input and output variables, respectively,
- $\varphi$ is a predicate over $X^I$ called input assumption, and
- $\psi$ is a predicate over $X^O$ called output guarantee.

We require that an interface accepts at least one input value and produces at least one output value.

Definition 2 (Well-Formedness). A stateless interface $M = \langle X^I, X^O, \varphi, \psi \rangle$ is well-formed if

1. $\varphi$ is satisfiable and
2. $\psi$ is satisfiable.
Refinement.

The refinement relation for interfaces is such that whenever an interface \( N \) refines an interface \( M \), then \( M \) can be replaced by \( N \) in every design that provides inputs satisfying the assumption of \( M \) and expects outputs satisfying the guarantee of \( M \). Hence, the refining interface \( N \) has to accept at least the same inputs as \( M \), and may produce a subset of the possible outputs of \( M \).

**Definition 3.** Given two well-formed stateless interfaces \( M = (X_M^I, X_M^O, \varphi_M, \psi_M) \) and \( N = (X_N^I, X_N^O, \varphi_N, \psi_N) \), we say that \( N \) refines \( M \), written \( N \preceq M \), if

1. \( X_M^I \cup X_N^I \cap (X_M^O \cup X_N^O) = \emptyset \),
2. \( \varphi_M \rightarrow \varphi_N \), and
3. \( \psi_N \rightarrow \psi_M \).

The classical theory of interfaces [4, 8] includes two operations to compose stateless interfaces:

- The connection allows to connect output variables to input variables of an interface.
- The parallel composition describes how to obtain the interface of a component that combines two or more sub-components running in parallel.

We recall the definition and properties of these two operations, and we introduce the new shared refinement, a binary operation that gives the most general interface refining two given interfaces.

Connection.

A connection consists of a set of pairs of variables defining which variables are connected when the connection is applied to an interface. The first component of each pair in a connection is an output variable, and the second an input variable to which the output is connected.

**Definition 4 (Connection).** A connection \( \theta \) is a set of pairs \((x, y)\) of variables, consisting of a source variable \( x \) and a target variable \( y \), such that for all pairs \((x, y), (x', y) \in \theta\), if \( x \neq x' \), then \( y \neq y' \).

We denote the set of source variables of \( \theta \) by \( S_\theta \) and the set of target variables by \( T_\theta \). The predicate \( \rho_\theta \) denotes \( \bigwedge_{(x, y) \in \theta} (x = y) \).

A connection is compatible with an interface \( M \), if (1) its source variables are all output variables of \( M \), (2) its target variables are all input variables of \( M \), and (3) when the source variables are connected to the target variables (i.e., \( \rho_\theta \) holds), there exist values of the remaining input variables of \( M \) such that the assumption of \( M \) is satisfied for all values of the output variables of \( M \) that satisfy the guarantee of \( M \).

**Definition 5 (Compatibility for Connection).** A stateless interface \( M = (X^I, X^O, \varphi, \psi) \) is compatible with a connection \( \theta \) if the following conditions hold:

1. \( S_\theta \subseteq X^O \),
2. \( T_\theta \subseteq X^I \), and
3. the predicate \( \hat{\varphi} \equiv \forall X^O \cdot \forall T_\theta \cdot (\psi \land \rho_\theta) \rightarrow \varphi \) is satisfiable.

If \( M \) is compatible with \( \theta \), then the result of applying \( \theta \) to \( M \) is the stateless interface \( M\theta = (\hat{X}^I, \hat{X}^O, \hat{\varphi}, \hat{\psi}) \), where

- \( \hat{X}^I = X^I \setminus T_\theta \),
- \( \hat{X}^O = X^O \cup T_\theta \), and
- \( \hat{\psi} \equiv (\psi \land \rho_\theta) \).

**Theorem 1.** Let \( M \) be a well-formed stateless interface and \( \theta \) be a connection. If \( M \) is compatible with \( \theta \), then \( M\theta \) is a well-formed stateless interface.

The connection supports independent implementability. Given a design \( M\theta \) and an interface \( N \) that refines \( M \), we can replace \( M \) by \( N \) in the design \( M\theta \), independently of \( \theta \), because \( N\theta \) refines \( M\theta \).

**Theorem 2.** Let \( M \) and \( N \) be two well-formed stateless interfaces and \( \theta \) be a connection. If \( N \preceq M \) and \( M \) is compatible with \( \theta \), then \( N \) is compatible with \( \theta \) and \( N\theta \preceq M\theta \).

Parallel composition.

Parallel composition allows to combine interfaces that are compatible with each other. Two interfaces are compatible for parallel composition if (1) the sets of output variables are compatible with each other. Two interfaces are compatible for parallel composition if (1) the sets of output variables are disjoint, (2) the input variables of each interface are disjoint from the output variables of the other interface, and (3) the conjunction of the guarantees is satisfiable.

**Definition 6 (Parallel Composition).** Two stateless interfaces \( M = (X_M^I, X_M^O, \varphi_M, \psi_M) \) and \( N = (X_N^I, X_N^O, \varphi_N, \psi_N) \) are compatible for parallel composition, written \( M \parallel N \), if

1. \( X_M^O \cap X_N^O = \emptyset \),
2. \( X_M^I \cap X_N^I = \emptyset \), \( X_M^O \cap X_N^O = \emptyset \), and
3. \( \varphi_M \land \varphi_N \) is satisfiable.

If \( M \parallel N \) holds, then the parallel composition is the stateless interface \( M \parallel N = (X^I, X^O, \hat{\varphi}, \hat{\psi}) \), where

- \( \hat{X}^I = X_M^I \cup X_N^I \),
- \( \hat{X}^O = X_M^O \cup X_N^O \),
- \( \hat{\varphi} \equiv (\varphi_M \land \varphi_N) \), and
- \( \hat{\psi} \equiv (\psi_M \land \psi_N) \).

**Theorem 3.** Let \( M \) and \( N \) be two well-formed stateless interfaces. If \( M \parallel N \) holds, then \( M \parallel N \) is a well-formed stateless interface.

The parallel composition supports independent implementability. In a design \( M \parallel S \), we can replace \( M \) by any \( N \) that refines \( M \) if all variables common to \( N \) and \( S \) are also variables of \( M \), because then \( N \parallel S \) refines \( M \parallel S \). Intuitively, the new variables of \( N \) should not conflict with variables of the design \( S \).

**Theorem 4.** Let \( M, N, \) and \( S \) be three well-formed stateless interfaces such that \( X_N \cap X_S \subseteq X_M \). If \( M \parallel S \) and \( N \preceq M \), then \( N \parallel S \) and \( N \parallel S = (M \parallel S) \parallel S \).
Shared refinement.

The shared refinement allows to describe the interface of a component that is meant to work in two or more environments based on separate descriptions of each environment. Note that different environment descriptions may use different variable names. Since it is the choice of the user to decide which variables are going to be shared, we assume that the interface variables are renamed accordingly before the shared refinement is applied.

The shared refinement of two interfaces needs to be able to replace each of the given interfaces. Therefore, it has to refine both interfaces. In order to ensure that the combined interface is well-formed, we introduce the notion of shared refinability. Two interfaces are shared refinable if (1) the input variables do not overlap with the output variables and (2) their output guarantees do not contradict each other.

Definition 7 (Shared refinement). Two stateless interfaces \( M = (X_M^I, X_M^O, \varphi_M, \psi_M) \) and \( N = (X_N^I, X_N^O, \varphi_N, \psi_N) \) are shared refinable, written \( M \sim N \), if

1. \( (X_M^I \cup X_N^I) \cap (X_M^O \cup X_N^O) = \emptyset \)
2. \( \psi_M \land \psi_N \) is satisfiable.

If \( M \sim N \) holds, then the shared refinement of \( M \) and \( N \) is the interface \( M \cap N = (X^I, X^O, \hat{\varphi}, \hat{\psi}) \), where

- \( \hat{X}^I = X_M^I \cup X_N^I \),
- \( \hat{X}^O = X_M^O \cup X_N^O \),
- \( \hat{\varphi} \equiv (\varphi_M \lor \varphi_N) \), and
- \( \hat{\psi} \equiv (\psi_M \land \psi_N) \).

Note that the assumption of the shared refinement of two interfaces is the disjunction of their assumptions, while in the parallel composition the assumptions are conjoined.

Theorem 5. Let \( M \) and \( N \) be two well-formed stateless interfaces. If \( M \sim N \), then \( M \cap N \) is a well-formed stateless interface.

Proof. Let \( M = (X_M^I, X_M^O, \varphi_M, \psi_M) \) and \( N = (X_N^I, X_N^O, \varphi_N, \psi_N) \). Due to well-formedness of \( M \) and \( N \), we have that \( \varphi_M \) and \( \varphi_N \) are satisfiable, and therefore \( \hat{\varphi} \equiv \varphi_M \lor \varphi_N \) is satisfiable. It follows from \( M \sim N \) that \( \forall \psi \equiv (\psi_M \land \psi_N) \) is satisfiable.

Example 1. Assume that after decomposing a design, we obtain (among others) two components with interface descriptions \( M \) and \( N \), respectively. Both interfaces refer to input variable \( x \) and output variable \( y \). The interfaces are depicted in Figure 2. The interface \( M \) states that the component has to accept all even numbers, and produces numbers that are multiple of 3. The second interface \( N \) takes numbers greater than 0 and guarantees multiples of 4. A common implementation has to accept all even numbers and all numbers greater than 0. Furthermore, it has to guarantee that every output is a multiple of 3 and 4, therefore a multiple of 12.

In the following, we prove that the shared refinement of two interfaces subsume all behaviors of the given interfaces.

Theorem 6 (Greatest lower bound). Let \( M \) and \( N \) be two well-formed stateless interfaces. If \( M \sim N \), then \( M \cap N \preceq M \) and \( M \cap N \preceq N \), and for all well-formed stateless interfaces \( S \), if \( S \preceq M \) and \( S \preceq N \), then \( S \preceq M \cap N \).

Proof. Let \( M \cap N = (X^I, X^O, \hat{\varphi}, \hat{\psi}) \), let \( M = (X_M^I, X_M^O, \varphi_M, \psi_M) \), and let \( N = (X_N^I, X_N^O, \varphi_N, \psi_N) \).

First, we show that \( M \cap N \preceq M \). The proof of \( M \cap N \preceq N \) is analogous. From Condition (1) of Definition 7, since \( M \sim N \), we have that \( (X^I \cup X_N^I) \cap (X^O \cup X_N^O) = \emptyset \). Furthermore, \( \varphi_M \) implies \( \hat{\varphi} \equiv \varphi_M \lor \varphi_N \) and \( \forall \psi \equiv (\psi_M \land \psi_N) \) implies \( \psi_M \). Hence \( M \cap N \preceq M \).

Second, we show that every well-formed interface \( S = (X_S^I, X_S^O, \varphi_S, \psi_S) \) that refines both \( M \) and \( N \) is a refinement of \( M \cap N \). Since \( S \preceq M \) and \( S \preceq N \), we have that \( (X_S^I \cup X_M^I) \cap (X_S^O \cup X_M^O) = \emptyset \) and \( (X_S^I \cup X_N^I) \cap (X_S^O \cup X_N^O) = \emptyset \). From Condition 1 of Definition 7, since \( M \sim N \), we know that \( (X_M^I \cup X_N^I) \cap (X_M^O \cup X_N^O) = \emptyset \). disappears, so \( (X_S^I \cup X_M^I) \cap (X_S^O \cup X_M^O) = \emptyset \) holds. It follows from \( \varphi_M \rightarrow \varphi_S \) and \( \varphi_N \rightarrow \varphi_S \) that \( \forall \varphi \rightarrow \varphi_S \) (because \( \hat{\varphi} \equiv \varphi_M \lor \varphi_N \)). Furthermore, \( \forall \psi \rightarrow \psi_M \) and \( \forall \psi \rightarrow \psi_N \) implies \( \forall \psi \rightarrow \psi \) (because \( \forall \psi \equiv (\psi_M \land \psi_N) \)).

3. STATEFUL INTERFACES

We consider interfaces with an internal state, analogous to the Moore interfaces of [2]. In each state of the interface, an assumption predicate constrains the input variables, and a predicate over output variables provides an output guarantee. The state of the interface changes according to a deterministic transition function over input and output variables.

Definition 8 (Stateful interface). A stateful interface is a tuple \( M = (X^I, X^O, Q, \tilde{q}, \varphi, \psi, \rho) \), where

- \( X^I, X^O \) are disjoint sets of input and output variables.
- We define \( X_M = X^I \cup X^O \);
- \( Q \) is a finite set of locations (or states), and \( \tilde{q} \in Q \) is the initial location;
- \( \varphi \) and \( \psi \) are two labelings that associate with each location \( q \in Q \) an input assumption predicate \( \varphi(q) \) over \( X^I \), and an output guarantee predicate \( \psi(q) \) over \( X^O \);
- \( \rho \) is a labeling that associates with each pair of locations \( q, q' \in Q \) a predicate \( \rho(q,q') \) over \( X_M \), called the transition guard.
Example 2. We illustrate the stateful model of interfaces with the specification of a FIFO buffer. The buffer has two Boolean input variables \(\text{enq}\) and \(\text{deq}\), which are set to perform an enqueue or dequeue operation, and two Boolean output variables \(E\) and \(F\), which describe whether the buffer is empty or full. In Figure 3, we present a specification \(S_1\) for this buffer, which we assume to be of size 2. States are labeled by their guarantee, and transitions are labeled by their guards. The assumption in a state is the disjunction of the guards of the outgoing transitions. Initially, the buffer is empty \((\bar{E})\), and enqueuing makes it full. If the buffer is full \((E)\), then dequeueing makes it empty, and dequeueing makes it full. If the buffer is full \((E)\), then enqueuing is not allowed. Simultaneous enqueue and dequeue operations have no effect (but they are allowed).

Given a set \(X\) of variables, we denote by \(\mathcal{V}[X]\) the set of all valuations \(v\) for \(X\), i.e., the functions that assign to each \(x \in X\) a value \(v(x)\). Given a predicate \(\varphi\) on \(X\), we write \(v \models \varphi\) if the valuation \(v\) satisfies \(\varphi\).

An execution of \(M\) is a sequence \(q_0, v_0, q_1, \ldots, q_n, v_n, q_{n+1}\) of states \(q_k \in Q\) and valuations \(v_k \in \mathcal{V}[X_M]\) such that \(q_0 = \hat{q}\), and \(v_k \models \varphi_M(q_k) \land \psi_M(q_k) \land \rho_M(q_k, q_{k+1})\) for all \(0 \leq k \leq n\). We say that the sequence \(v_0, \ldots, v_n\) is a trace of \(M\), and that the states \(q_0, \ldots, q_{n+1}\) are reachable in \(M\). We denote by \(\text{Traces}(M)\) the set of all traces of \(M\), and by \(\text{Reach}(M)\) the set of all states that are reachable in \(M\).

Definition 9 (Well-formedness). A stateful interface \(M = (X^I, X^O, Q, \hat{q}, \varphi, \psi, \rho)\) is well-formed if for all states \(q \in \text{Reach}(M)\),

1. both \(\varphi(q)\) and \(\psi(q)\) are satisfiable,
2. \((\varphi(q) \land \psi(q)) \rightarrow \exists q': \rho(q, q')\) is valid, and
3. \((\rho(q, q') \land \rho(q, q'')) \rightarrow (q' = q'')\) is valid for all \(q', q'' \in Q\).

Well-formedness ensures that the interface is non-blocking by condition (1) and (2), and deterministic by condition (3).

Refinement and parallel composition.

The definition of refinement and parallel composition for stateful interfaces follows the lines of [2].

Definition 10 (Refinement). Given two stateful interfaces \(M = (X^I_M, X^O_M, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M)\) and \(N = (X^I_N, X^O_N, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N)\), we say that \(N\) refines \(M\), written \(N \preceq M\), if

1. \((X^I_M \cup X^I_N) \cap (X^O_N \cup X^O_M) = \emptyset\) and
2. there exists a relation \(R \subseteq Q_N \times Q_M\) such that \((\hat{q}_N, \hat{q}_M) \in R\), and for all pairs \((r, q) \in R\),
   - \(\psi_M(q) \rightarrow \varphi_N(r)\) is valid,
   - \(\psi_N(r) \rightarrow \psi_M(q)\) is valid, and
   - \(\rho_M(q, q') \land \rho_M(q, q'')\) and \(\rho_N(r, r')\) are valid, then \((r', q') \in R\).

Such a relation \(R\) is an alternating simulation relation [1] and we say that \(R\) is a witness for \(N \preceq M\).

For parallel composition, we do require that the two stateful interfaces share no output variable, but we allow variables that are both output variable in one interface and input variable in the other interface. This implicitly enables a connection operation for stateful interfaces.

Definition 11 (Parallel composition). Given two stateful interfaces \(M = (X^I_M, X^O_M, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M)\) and \(N = (X^I_N, X^O_N, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N)\), let

- \(X^O_M = X^O_M \cup X^O_N\),
- \(X^I_M = (X^I_M \cup X^I_N) \setminus X^O_P\),
- \(Q_P = Q_M \times Q_N\), and
- \(\hat{q}_P = (\hat{q}_M, \hat{q}_N)\),

and for all states \(q, q' \in Q_M\) and \(r, r' \in Q_N\), let

- \(\psi_P(q, r) \equiv (\psi_M(q) \land \psi_N(r))\) and
- \(\rho_P((q, r), (q', r')) \equiv (\rho_M(q, q') \land \rho_N(r, r'))\).

We say that \(M\) and \(N\) are compatible for parallel composition, written \(M \parallel N\), if

1. \((X^I_M \cup X^I_N) \cap (X^O_N \cup X^O_M) = \emptyset\) and
2. there exists a labeling \(\varphi_P\) that associates with each pair \((q, r) \in Q_M \times Q_N\) a predicate \(\varphi_P(q, r)\) on \(X^I_P\), such that
   - \(\varphi_P(q, r)\) is satisfiable, and
   - for all executions \((q_0, r_0), (q_1, r_1), \ldots, (q_n, r_n)\) of \((X^I_P, X^O_P, Q_P, \hat{q}_P, \varphi_P, \psi_P, \rho_P)\), we have that \(v_k \models (\varphi_M(q_k) \land \varphi_N(r_k))\) holds for all \(0 \leq k < n\).
Weakest labeling.

of an interface, it is easy to show that there always exists a

\( \varphi \)

ditions (2)

parallel composition is checked by solving a safety game. The

are not reachable in

\( \varphi \)

weakest labeling.

if all variables common to \( N \) and \( S \) are also

variables of \( M \), because then \( N \parallel S \) refines \( M \parallel S \).

\textbf{Theorem 7.} [2] Let \( M, N \), and \( S \) be three well-formed

stateful interfaces such that \( X_N \cap X_S \subseteq X_M \). If \( M \approx S \) and

\( N \leq M \), then \( N \approx S \) and \( N \parallel S \leq M \parallel S \).

\textbf{Shared refinement.}

The shared refinement \( M \cap N \) is the weakest interface that

refines both \( M \) and \( N \). Roughly, the interface for \( M \cap N \)

starts with a classical product construction of \( M \) and \( N \),

where the assumptions allow inputs that satisfy the assumption

of either \( M \) or \( N \). As long as both assumptions of \( M \)

and \( N \) are satisfied by the inputs, the outputs must satisfy

the conjunction of the guarantees of \( M \) and \( N \). When the assumption of \( M \) (resp. \( N \)) is violated, then the interface

jumps to a copy of \( N \) (resp. \( M \)), where the assumptions and

guarantees are those of \( N \) (resp. \( M \)). Note that the interface
does not allow inputs that violate the assumptions of both

\( M \) and \( N \).

\textbf{Definition 12 (shared refinement).} Given two

stateful interfaces \( M = (X_M, X_M^O, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M) \)

and \( N = (X_N, X_N^O, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N) \), let \( P \) be the interface

\( (X_P, X_P^O, Q_P, \hat{q}_P, \varphi_P, \psi_P, \rho_P) \) where

- \( X_P^O = X_M^O \cup X_N^O \),
- \( X_P = X_M \cup X_N \),
- \( Q_P = (Q_M \times Q_N) \cup Q_M \cup Q_N \),
- \( \hat{q}_P = (\hat{q}_M, \hat{q}_N) \),
- \( \varphi_P \) and \( \psi_P \) are defined, for all \( q \in Q_M \) and \( r \in Q_N \), by
  \( \varphi_P(q, r) \equiv (\varphi_M(q) \lor \varphi_N(r)) \)
  \( \psi_P(q, r) \equiv (\psi_M(q) \land \psi_N(r)) \)
- \( \varphi_P(q) \equiv \varphi_M(q) \lor \varphi_N(q) \land \psi_P(r) \equiv \psi_M(q) \land \psi_N(q) \),
- \( \rho_P(q, q') \equiv \rho_M(q, q') \lor \rho_N(q, q') \land \rho_P(r, r') \equiv \rho_M(r, r') \lor \rho_N(r, r') \),

We say that \( M \) and \( N \) are shared refinable, written \( M \sim N \), if

- \( X_P^O \cap X_P^O = \emptyset \) and
- \( \psi_P(p) \) is satisfiable for all states \( p \in \text{Reach}(P) \).

When \( M \sim N \), the shared refinement \( M \cap N \) is the well-

formed stateful interface \( P \).
First, we show that the shared refinement has the flavor of a greatest lower bound operator.

Let $M$ and $N$ be two well-formed stateful interfaces. If $M \sim N$, then $M \cap N \preceq M$ and $M \cap N \preceq N$, and for all well-formed stateful interfaces $S$, if $S \preceq M$ and $S \preceq N$ then $S \preceq M \cap N$.

Proof. Let $M$ and $N$ be stateful interfaces such that $M \sim N$ and let $M \cap N = \langle X_F^M, X_O^M, Q_P, \hat{q}_P, \hat{q}_S, \hat{q}_M, \psi_P, \psi_S, \psi_M \rangle$. First, we show that $M \cap N \preceq M$. We have $X_F^M \cap X_F^N = \emptyset$ and thus $\langle X_F^M \cup X_F^N, X_O^M \cup X_O^N, Q_P, \hat{q}_P, \hat{q}_S, \hat{q}_M, \psi_P, \psi_S, \psi_M \rangle = \emptyset$. Let $R \subseteq Q_P \times Q_M$ be the union of $R_1 = \{(q,r) \mid (q,r) \in Q_P \}$ and $R_2 = \{(q,r) \mid q \in Q_M\}$. We have $(\hat{q}_P, \hat{q}_M) \in R$ for all $(q,r) \in R_1$.

(1) $\varphi_M(q) \rightarrow \varphi_P(q,r)$ since $\varphi_P(q,r) \equiv \varphi_M(q) \lor \varphi_N(r)$,

(2) $\psi_P(q, r) \rightarrow \psi_M(q)$ since $\psi_P(q, r) \equiv \psi_M(q) \land \psi_N(r)$,

(3) if $\varphi_M(q) \land \psi_P(q, r) \land \rho_P(q, r) \land \rho_M(q, q')$, then either $q' = q$, and then $(q', q') \in R_2$, or $q' = (q', r')$ for some $r' \in Q_N$ and then $(q', q') \in R_1$.

Analogous results trivially hold for $R_2$, and therefore $R$ is a witness for $M \cap N \preceq M$. We have $M \cap N \preceq N$ by a symmetrical argument.

Second, we show that $S \preceq M$ and $S \preceq N$ implies $S \preceq M \cap N$ for all $S$. Since $X_F^S \cap X_F^M = \emptyset$, it is easy to show that $(X_F^S \cup X_F^M) \cap (X_O^S \cup X_O^M) = \emptyset$. Let $R_1$ be a witness for $S \preceq M$, and $R_2$ be a witness for $S \preceq N$. Let $R \subseteq Q_S \times Q_P$ be the union of $R_1 \cup R_2$, and $R_3 = \{(s, q, r) \mid (s, q) \in R_1 \text{ and } (s, r) \in R_2\}$. We have $(\hat{q}_S, \hat{q}_P) \in R$ because $(\hat{q}_S, \hat{q}_M) \in R_1$, and $(\hat{q}_S, \hat{q}_N) \in R_2$. Moreover, for all $(s, (q, r)) \in R_3$, we have

(1) $\varphi_P(q, r) \rightarrow \varphi_S(s)$ since $\varphi_P(q, r) \equiv \varphi_M(q) \lor \varphi_N(r)$, $\varphi_M(q) \rightarrow \varphi_S(s)$, and $\varphi_N(r) \rightarrow \varphi_S(s)$,

(2) $\psi_S(s) \rightarrow \psi_P(q, r)$ since $\psi_S(s) \rightarrow \psi_M(q)$, $\psi_S(s) \rightarrow \psi_N(r)$, and $\psi_P(q, r) \rightarrow \psi_S(s)$, and

(3) if $\varphi_P(q, r) \land \psi_S(s) \land \rho_S(s, s') \land \rho_P(q, r, p')$, then either

(a) $p' = (q', r')$ and $\varphi_M(q) \land \varphi_N(r) \land \rho_M(q, q') \land \rho_N(r, r')$, and then $\varphi_M(q) \land \psi_S(s) \land \rho_S(s, s') \land \rho_M(q, q')$ so that $(s', q') \in R_1$, and symmetrically $(s', r') \in R_2$, and therefore $(s', p') \in R_3$, or

(b) $p' = q' \in Q_M$ and $\varphi_M(q) \land \rho_M(q, p')$ and then $(s', p') \in R_1$, or

(c) $p' = r' \in Q_N$ and $(s', p') \in R_1$ by a symmetric argument.

Analogous results trivially hold for all $(s, p) \in R_1 \cup R_2$, and therefore $R$ is a witness for $S \preceq M \cap N$.

Finally, we have the following associativity property of shared refinement, which follows from the greatest lower bound property of Theorem 8.

THEOREM 9. Given three well-formed stateful interfaces $M$, $N$, and $S$, either both $(M \cap N) \cap S$ and $M \cap (N \cap S)$ are undefined, or they are both defined, and then they refine each other.

4. REUSE OF PCI DEVICES

We illustrate the use of shared refinement for specifications of the peripheral interconnection of components on a bus. On the one hand, for the PCI bus, we consider the functional specification as described in [2] and the power management interface [9]. On the other hand, we consider the functional specification of a different peripheral bus. Finally, we show that the shared refinement of these three specifications is an interface of any implementation of a device that satisfies the power management specification, and is functionally compatible with both busses.

**Power management interface.**

A PCI function is a device that can be connected to the PCI bus, and which has to implement its own power management interface. According to the PCI Bus Power Management Interface Specification [9], each PCI function can be of one of four power-management states: D0, D1, D2, or D3 in decreasing order of power consumption. The states D0 and D3 are further split into D0hot, D0cold, D3hot, D3cold. To comply with the PCI standard, all PCI functions have to support the D0 and D3 states.
In Figure 6, we present a stateful interface $P$ for controlling the different power states for the PCI bus. The input signals are $X^I = \{\text{init}, \text{rst}, \text{pmc}, \text{gnt}, \text{off}\}$, and there is no output signal. In state $D_{\text{uninit}}$ the PCI function must be initialized by the system software (variable init) in order to be put in the active state $D_{\text{active}}$. Functions in $D_{\text{hot}}$ can move to the $D_{\text{active}}$ state and back via software by writing to the function’s PMCSR register (variable pmc). The difference between the two $D_{3}$ states is defined by the absence ($D_{\text{cold}}$) or presence ($D_{\text{hot}}$) of voltage $V_{cc}$ (regulated by the off signal). A PCI function can be transitioned into $D_{\text{cold}}$ states either by software (variable off) or by physically removing the power from its host PCI device. Functions in $D_{\text{cold}}$ can only get to $D_{\text{uninit}}$ by reapplying $V_{cc}$ and asserting the reset signal (rst) to the function’s host PCI device.

Additionally, the device may receive a signal gnt when it has been granted to access the PCI bus, but this should not happen when it is in one of the low-consumption states $D_{3}$.

**Bus request management.**

We consider the functional specification described in [2] (see Figure 7) for connection with the PCI bus, and we define a different specification for a different peripheral bus (see Figure 8). $F_1$ and $F_2$ are stateful interfaces of the device that can be attached to the corresponding peripheral bus. Interfaces $F_1$ and $F_2$ have the three states NotOwner, Request, and Owner (of the device). The input variable is gnt, and the output variable is req to request the bus.

As required by the PCI bus, the stateful interface $F_1$ specifies that the device either keeps requesting the bus until it is granted, or it withdraws the request and goes back to the NotOwner state. Further, the stateful interface $F_2$ gives a specification of a different bus, where the device is expected to send a req signal once and then wait to be granted.

Note that the two specifications $F_1$ and $F_2$ do not refine each other: the trace starting with req $\land$ $\neg$gnt, req $\land$ gnt, req $\land$ gnt, ... violates the assumption $\neg$gnt of the state NotOwner in interface $F_2$, while it can be continued according to the specification $F_1$. Therefore the device should recognize that it has to continue the execution according to the specification $F_1$. Further, the trace starting with req $\land$ $\neg$gnt, $\neg$req $\land$ gnt, $\neg$req $\land$ gnt, ... is allowed by specification $F_2$ but not by $F_1$.

Now we consider the interfaces $C_1 = P \cap F_1$ and $C_2 = P \cap F_2$. Any component that refines $C_1$ satisfies the power management specification $P$, and at the same time, such a component is compatible with a PCI bus as described with the functional interface $F_1$. An analogous statement holds for $C_2$. However, in order to enable the reuse of components, one may require that the same component refines both $C_1$ and $C_2$, or equivalently refines $C_1 \cap C_2 = P \cap F_1 \cap F_2$. Moreover, any component which refines both interfaces $C_1$ and $C_2$ has to refine $S = P \cap F_1 \cap F_2$, as it is the weakest interface with this property. Note that $C_1$ and $C_2$ are shared refinable, because $P$, $F_1$, and $F_2$ have trivial guarantees. Therefore, we can construct the stateful interface $S$, which has $5 \cdot 4 \cdot 4 - 1 = 79$ states.

The device implementations that are compliant with both busses are exactly those that refine $S$. The interface $S$ ensures that the guarantees of all three specifications $P$, $F_1$, and $F_2$ are satisfied as long as the assumptions of the three specifications are met. If the inputs no longer conform to the power specification $P$, then only the guarantees of the functional specifications $F_1$ and $F_2$ can be maintained. If the inputs no longer comply with the PCI specification $F_1$, then only the guarantees for $F_2$ and $P$ can be ensured. A similar statement holds for $F_2$. Finally, if the inputs violate the assumptions of two of the three specifications, then $S$ moves to a copy of the third interface and behaves according to that interface.

### 5. DISCUSSION

The shared refinement of two interfaces represents the most permissive interface that refines both interfaces. This can be viewed as a greatest lower bound property for the refinement relation (Theorem 8), which is defined as alternating simulation. Classically, the parallel composition also defines a greatest lower bound, but for the trace inclusion relation. For instance, in automata theory, the language $L(A\parallel B)$ of the parallel product of two automata is exactly the set of all traces that are common to the languages of $A$ and $B$, i.e., $L(A\parallel B) = L(A) \cap L(B)$. Notice that $\cap$ is the greatest lower bound for set inclusion.

In the theory of interfaces, the set of traces of the parallel composition $M\parallel N$ may not be the intersection of the traces of $M$ and $N$. More precisely, there exist two well-formed interfaces $M$ over variables $X_M$ and $N$ over $X_N$
that are compatible for parallel composition and such that \( \text{Traces}(M||N) \subseteq \text{Traces}'(M) \cap \text{Traces}'(N) \), where \( \text{Traces}'(M) \) (resp. \( \text{Traces}'(N) \)) is the set of sequences \( v_1, \ldots, v_n \) of valuations for \( X_M \cup X_N \) that agree with a trace of \( M \) (resp. \( N \)) on variables in \( X_M \) (resp. \( X_N \)). Consider the interfaces \( M \) and \( N \) in Figure 9 over the Boolean variables \( x \), \( y \), and \( z \), where \( X_N^1 = X_N^2 = \{ x \} \), \( X_N^3 = \{ y \} \), and \( X_N^4 = \{ z \} \). Assumptions and guarantees in all states are trivial except in \( q_2 \) and \( q_3 \), where the assumptions are \( \varphi_M(q_2) \equiv x \) and \( \varphi_N(q_3) \equiv \neg x \). In the parallel composition \( M||N \) (see Figure 10), the pair of states \((q_1, q'_1)\) and \((q_2, q'_2)\) is strengthened to \( \neg x \) in order to avoid a transition to \((q_2, q'_2)\). So traces starting with valuation \( v \) such that \( v(x) = \top \) and \( v(y) = v(z) = \bot \) are not included in \( \text{Traces}(M||N) \). On the other hand, \( M \) and \( N \) allow both all traces starting with valuations \( v_1 \) and \( v_2 \), respectively, such that \( v_1(x) = v_2(x) = \top \) and \( v_1(y) = v_2(z) = \bot \).

We note that interfaces strictly separate input and output variables, in the sense that assumptions refer to input variables only and guarantees to output variables only. This strict separation may force assumptions in the parallel composition to be stronger than intuitively necessary. In the previous example, one may expect the assumption of \((q_1, q'_1)\) to be \( \top \) and its guarantee to be \( \neg x \lor \neg y \lor \neg z \). This requires that guarantee predicates may refer to both input and output variables. Let us try to consider in the stateless case such an extended setting, which we call extended interfaces.

Note that assumptions would not be necessary anymore, as one can define the assumption \( \varphi \) as \( \exists X^O \cdot \psi \), i.e., the allowed values of the input variables are those for which the guarantee predicate is satisfiable. So every pair of an assumption and a guarantee can be written as a single (maybe different) guarantee that would describe the same interface. The well-formedness condition (analogous of Definition 2) then would simply require that \( \varphi \) be satisfiable (which implies that \( \psi \) is satisfiable). However, in the following we keep assumptions and guarantees separate, because it is the natural way to think about interfaces.

If we consider extended interfaces, the definition of shared refinement could be adapted to keep the greatest lower bound property. Indeed, with the extended guarantees, we could define more permissive interfaces that refine two given interfaces. Specifically, using the notation of Definition 12, the assumption \( \hat{\varphi} \) and the guarantee \( \hat{\psi} \) of the shared refinement \( M \sqcap N \) could be defined as

\[
\hat{\varphi} \equiv (\varphi_M \lor \varphi_N) \land \exists X^O \cdot \psi,
\]

- \( \hat{\psi} \equiv (\varphi_M \rightarrow \psi_M) \land (\varphi_N \rightarrow \psi_N) \).

The refined guarantee allows the new shared refinement to refine both \( M \) and \( N \) while the shared refinement of Definition 12 refines \( M \sqcap N \). Theorem 6 holds for extended interfaces with this new definition.

**Example 4.** Consider the interfaces \( M \) and \( N \) of Example 1, which are now considered to be extended interfaces. The new guarantee of \( M||N \) would be \((\text{even}(x) \rightarrow y \mod 3 = 0) \land (x > 0 \rightarrow y \mod 4 = 0)\), which is strictly weaker than the guarantee \( y \mod 12 = 0 \) in the original setting. For instance, \( y \) would not required to be a multiple of 12 when \( x \) is a positive odd number.

Extended interfaces seem to provide a stronger framework than classical interface theories. Unfortunately, the basic properties of stepwise refinement and independent implementability do not hold in the extended framework. Formally, the extended interfaces do not support congruence with connection, i.e., there exist a connection \( \theta \) and two well-formed extended interfaces \( M \) and \( N \) such that \( N \sqsubseteq M \) and \( M \) is compatible with \( \theta \), but \( N \) is not compatible with \( \theta \). Consider the stateless interfaces \( M \) and \( N \) over input variable \( x \) and output variable \( y \) with trivial assumptions and guarantees, except the guarantee of \( N \), which is \( \psi_N \equiv (y \neq x) \). Let \( \theta \) be the connection \( \{(y, x)\} \). The interface \( M \) is compatible with \( \theta \), and the interface \( M\theta \) has the trivial assumption and guarantee \( \psi_M \equiv (y = x) \). However, even though \( N \) refines \( M \), it is not compatible with \( \theta \) because the predicate \( \rho_\theta \equiv (y = x) \) contradicts the guarantee \( \psi_N \) and thus \( N\theta \) would not be well-formed owing to an unsatisfiable guarantee. Hence, the analogous of Theorem 2 for extended interfaces does not hold.

We believe that the well-formedness requirement that the guarantee must be satisfiable should not be dropped in an interface theory. Well-formed interfaces should have at least one environment in which they can be embedded. In fact, by definition an interface is well-formed iff it can be used in some context [5]. On the other hand, independent implementability formalized by congruence is a crucial aspect of the theory and should not be dropped either. It turns out that interface theories in which inputs and outputs are separated are the most general known framework in which these two features coexist.
Acknowledgments

We gratefully acknowledge discussions with Albert Benveniste, who convinced us that the operation $\cap$ is missing from interface theories and inspired us to start this work. As part of the same debt, we also acknowledge Albert’s coauthors of the talk on “Multiple viewpoints contracts and residuation,” which he gave at the 2008 Workshop on Foundations of Interface Technologies (FIT): Jean-Baptiste Raclet, Eric Badouel, Benoit Caillaud, and Roberto Passerone.

6. REFERENCES


