Mediator Synthesis in a Component Algebra with Data

Lukáš Holík[1], Malte Isberner[2], and Bengt Jonsson[3]

[1] Brno University of Technology, Czech Republic
[2] Technical University of Dortmund, Germany
[3] Uppsala University, Sweden

Abstract. We formulate a compositional specification theory for components that interact by directed synchronous communication actions. The theory is an extension of interface automata which is also able to capture both absence of deadlock as well as constraints on data parameters in interactions. We define refinement, parallel composition, and quotient. The quotient is an adjoint of parallel composition, and produces the most general component that makes the components cooperate to satisfy a given system specification. We show how these operations can be used to synthesize mediators that allow components in networked systems to interoperate. This is illustrated by application to the synthesis of mediators in e-commerce applications.

1 Introduction

Modern software-intensive systems are increasingly composed of independently developed and network-connected software components. In many cases, these components exhibit heterogeneous behaviour, e.g., employing different protocols, which prevents them from cooperating to achieve user-level goals. Such cases call for the synthesis of *mediators*, which are intermediary software entities that allow software components to interact by coordinating their behaviours. Mediator synthesis has many different applications, including protocol converters [22], web service composition [4], and driver synthesis [20].

Component-based development, including mediator synthesis, should be performed within a specification theory. The theory should express how specifications capture the requirements for a component to function in an intended system context, while operators and refinement relations allow the composition and comparison of specifications, in analogy with how components are composed and refined towards an overall system design. Several such theories have been proposed, one of the earliest by Olderog and Hoare for Hoare-style specifications of Communicating Sequential Processes [14]. A more recent theory is that of interface automata of de Alfaro and Henzinger [7], in which components are assumed to communicate by synchronisation of input and output (I/O) actions, with the understanding that outputs are non-blocking. If an output is issued when another component is unwilling to receive it, a communication mismatch is said to occur. This allows to capture assumptions on the behaviour of a component's environment.

The theory of interface automata is formulated in a finite-state, finite-alphabet setting. This falls short of adequately capturing the behavior of realistic systems, where communicating components exchange messages containing data values (e.g., header information or payload) ranging over large or infinite domains, and the overall control flow depends crucially on these values. Furthermore, the theory does not have any facilities for modeling progress or deadlock properties.

In this paper, we address the aforementioned issues by extending the formalism of interface automata to model both progress and constraints on data parameters. The constraints on data parameters are restricted to equalities and negated equalities between data values; this still allows to use the formalism in many interesting applications. We define refinement, *parallel composition* for structural composition of components, and *quotient* for synthesizing new components to satisfy partial requirements. The quotient is an adjoint of parallel composition, and produces the most general component that makes the system components cooperate to satisfy a given system specification.

Thereafter, we demonstrate how our specification theory and its operators can be used to automatically synthesize mediators in component-based systems. Several previous approaches to mediator synthesis require the user to supply a specification of the overall system functionality. However, the facilities of our theory allow to automatically generate such a system specification in many situations. More precisely, the ability to model constraints on data allows to capture data flow properties of a system, which in turn induces constraints on the control flow that result in a specification. We further show how the ability to specify progress properties results in the pruning of unproductive behavior from the resulting mediator. We have implemented the computation of quotients, and show the applicability of our approach by synthesizing mediators in e-commerce applications.

Related Work Our component algebra is rooted in the theory of interface automata, which was developed for finite-state specifications, without covering data or progress [7]. The extension to progress was developed in our previous work [6]. Another related theory is the trace theory of Dill [8], which is based on the same distinction between inputs and outputs, and a notion of progress based on infinite traces. Another variation of this theory is the modal interfaces by Raclet and others [15–18], and the modal specifications by Larsen et al. [12], where *must* and *may* transitions play the role of input and output transitions. Yet another related theory is the ioco theory for model based testing due to Tretmans [21], which has a notion of deadlock that is very similar to ours.

For finite-state specifications, several works have presented a construction for computing quotients. Bhaduri and Ramesh [5] showed that for finite-state interface automata, which do not capture progress properties, quotient can be reformulated by combining parallel composition and renaming of interface actions: this reformulation does not hold when deadlock and progress are considered. In our previous work [2], we used finite-state quotient to synthesize mediators. In the absence of facilities for modeling data, it was necessary to use an ontology for correlating different actions. In the current paper, the ability to specify constraints on data makes the use of ontologies unnecessary.

All works mentioned in the preceding paragraph are developed for the finite-state case, not covering data. There is one work that extends the theory of modal specifications with data [1]. It is based on a different model of interaction between processes, where data values are communicated via shared variables rather than as parameters in synchronization primitives. Moreover, the theory of modal specifications is based on a semantic foundation different from ours, in which specifications are compared in terms of the sets of transition systems that implement them. Our theory is based on notions of simulation between specifications, which allows mainstream techniques from controller synthesis to be adapted to the extension with data.

The synthesis of mediators has been addressed in works that target web service composition, e.g., in [3, 4, 9, 10]. The authors of these works model components, representing web services, using automata extended with data. Mediators are synthesized using planning techniques, typically by a forward search in the space of possible mediators. The synthesized mediators are not required to be "best" or "most general" in a sense provided by some speficiation theory. In [4], loop-free controllers are synthesized that guarantee that the composed system reaches a target state while avoiding unsafe states. In contrast, our specification is a combination of a safety condition and a deadlock-freeness condition; these two kinds of specifications are of incomparable expressive power. Further difference include that our synthesized controllers may contain loops, and [4] under-approximates infinite data domains by finite ones.

2 Processes

In this section, we define our model of processs with data.

We assume an unbounded domain $\mathcal{D}$ of *data values* and a set of *actions*. Each action has a certain *arity*, which is the number of data parameters it takes. A *symbol* is a term of the form $\alpha(d_1, \ldots, d_n)$, where α is an action of arity n, and $d_1, \ldots, d_n$ are data values in $\mathcal{D}$. We further assume a set of *variables*, ranged over by x, y, z, etc., and a set of *formal parameters* ranged over by p_i for $i \in \mathbb{N}$. A *parameterized symbol* is of form $\alpha(p_1, \ldots, p_n)$, where α is an action of arity n, and $p_1, \ldots, p_n$ are distinct formal parameters. A *guard* is a conjunction of equalities and negated equalities over variables and formal parameters.

Definition 1. A *process* $\mathcal{P}$ is a tuple $\mathcal{P} = (I_\mathcal{P}, O_\mathcal{P}, L_\mathcal{P}, \ell^0_\mathcal{P}, X_\mathcal{P}, \sigma^0_\mathcal{P}, \delta_\mathcal{P}, \mathit{Quie}_\mathcal{P})$, where

- $I_\mathcal{P}$ is a finite set of *input actions*,
- $O_\mathcal{P}$ is a finite set of *output actions*,
- $L_\mathcal{P}$ is a finite set of *locations*,
- $\ell^0_\mathcal{P} \in L_\mathcal{P}$ is the *initial location*,
- $X_\mathcal{P}$ is a tuple $x_1, \ldots, x_k$ of *variables*,
- $\sigma^0_\mathcal{P} \colon X_\mathcal{P} \to \mathcal{D}$ is the *initial valuation*, assigning to each variable in $X_\mathcal{P}$ an initial value,
- $\delta_\mathcal{P}$ is a finite set of *transitions*, each of which is of form $\langle \ell, \mathit{stmt}, \ell' \rangle$ where $\ell, \ell' \in L_\mathcal{P}$ are locations, and *stmt* is a statement of form

$$\alpha(p_1, \ldots, p_n) \ ; \ g \ ; \ x_1, \ldots, x_k := e_1, \ldots, e_k \ ,$$

  where
  - $\alpha(p_1, \ldots, p_n)$ is a parameterized symbol with $\alpha \in (I_\mathcal{P} \cup O_\mathcal{P})$,
  - g is a guard over $X_\mathcal{P}$ and $p_1, \ldots, p_n$,

    - $x_1, \ldots, x_k := e_1, \ldots, e_k$ assigns to each variable $x_i \in X_\mathcal{P}$ an expression e_i over variables in $X_\mathcal{P}$ and formal parameters in $p_1, \ldots, p_n$.
- $Quie_\mathcal{P}$ maps each location in $L_\mathcal{P}$ to a predicate over $X_\mathcal{P}$. □

We write $\bar{p}$ for $p_1, \ldots, p_n$, $\bar{d}$ for $d_1, \ldots, d_n$, $\overline{x}$ for $x_1, \ldots, x_k$, and similarly for $\overline{e}$. We write $A_\mathcal{P}$ for $I_\mathcal{P} \cup O_\mathcal{P}$. We use the term *$\alpha$-transition* for a transition $\langle \ell,\ \alpha(\bar{p}); g; \overline{x} := \overline{e}\ , \ell' \rangle$ with the action α. For an action α and a location $\ell \in L_P$, let g_ℓ^α denote the disjunction of the guards of the outgoing α-transitions from ℓ; if there is no such statement, then g_ℓ^α is defined as *false*.

Intuitively, a process is at any point in time in a state, given by a location and a valuation of its variables. The process can change its state by performing a transition $\langle \ell,\ \alpha(\bar{p}); g; \overline{x} := \overline{e}\ , \ell' \rangle$, provided that its current location is ℓ, and that there is some $\alpha(\bar{d})$ that makes the guard $g[\bar{d}/\bar{p}]$ evaluate to true under the current valuation. It synchronizes via the symbol $\alpha(\bar{d})$, binds the formal parameters $\bar{p}$ to data values $\bar{d}$, and simultaneously assigns new values to the variables according to $\overline{x} := \overline{e}$.

The distinction between input and output actions is interpreted as follows. When in a state $\langle \ell, \sigma \rangle$, a process $\mathcal{P}$ is willing to receive any input $\alpha(\overline{d})$ that is enabled in the current state. Some input symbols may not be enabled: this expresses the assumption that the environment never generates these inputs when $\mathcal{P}$ is in $\langle \ell, \sigma \rangle$. This is in contrast to I/O-automata [11, 13], which in every state must be prepared to receive any input. A process $\mathcal{P}$ can also emit any output symbol that is enabled. The predicate $Quie_\mathcal{P}(\ell)$ constrains when $\mathcal{P}$ can be *quiescent*, in the sense that if no input is received, the system *must* eventually emit some output symbol, *unless* the predicate $Quie_\mathcal{P}(\ell)$ is satisfied under the current valuation σ. Quiescence can be used to model termination or deadlock. We require that any process $\mathcal{P}$ is *consistent* with $Quie_\mathcal{P}$ in the sense that for any location ℓ of $\mathcal{P}$, the formula

$$Quie_\mathcal{P}(\ell) \vee \bigvee_{\alpha \in O_\mathcal{P}} \exists \overline{p}.\ g_\ell^\alpha$$

is universally valid. Here, $\overline{p}$ refers to the formal parameters of the respective action α.

Determinism We assume that processes are *deterministic*, i.e., if g_1 and g_2 are the guards of two outgoing α-statements from a location ℓ, then g_1 and g_2 are mutually exclusive. Note that this still allows to express non-determinism on the level of data values. Consider, for instance, a process which nondeterministically selects an identifier, assigns it to a variable id and then transmits it in an action *showid*(p), where p is id. We can represent this behavior by a single transition labeled by the statement

$$\textit{showid}(p)\ ;\ \textit{true}\ ;\ id := p\ \ .$$

Since this transition has the guard *true*, the parameter of *showid* can be an arbitrary identifier. We have just slightly remodeled the process so that id is assigned in connection with the transition, rather than before the transition; the external behavior remains the same. We use this modeling idiom in several of the examples in Section 6.

Underlying Theory In this paper, we have restricted the relations in guards and predicates to equalities and negated equalities, and avoided functions in expressions (i.e., each e_i in an assignment $\overline{x} := \overline{e}$ is either a formal parameter or a variable). Under these restrictions, the elimination of existential quantifiers is fairly simple (as described at the end of this section). Our solution to the synthesis problem does not inherently rely on these restrictions. In principle, the set of relations and functions could be extended to include other theories as well. Our constructions would still work under restrictions that guarantee termination of concerned algorithms. We leave the precise formulation of such restrictions as future work.

Semantics of Processes For a set X of variables, an X-*valuation* is a partial mapping from X to $\mathcal{D}$. For an X-valuation σ and a guard g over X, we let $\sigma \models g$ denote that g evaluates to true under the valuation σ. A *state* of $\mathcal{P}$ is a pair $\langle \ell, \sigma \rangle$ where $\ell \in L_{\mathcal{P}}$ and σ is an $X_{\mathcal{P}}$-valuation. The *initial state* of $\mathcal{P}$ is $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle$, where $\ell^0_{\mathcal{P}}$ is the initial location and $\sigma^0_{\mathcal{P}}$ is the initial valuation. A *step* of $\mathcal{P}$, denoted by $\langle \ell, \sigma \rangle \xrightarrow{\alpha(\bar{d})}_{\mathcal{P}} \langle \ell', \sigma' \rangle$, transfers $\mathcal{P}$ from $\langle \ell, \sigma \rangle$ to $\langle \ell', \sigma' \rangle$ while performing the symbol $\alpha(\bar{d})$. According to whether α is in $I_{\mathcal{P}}$ or $O_{\mathcal{P}}$, we call $\langle \ell', \sigma' \rangle$ an *input* or *output* successor of $\langle \ell, \sigma \rangle$. It is derived from a transition $\langle \ell, \mathit{stmt}, \ell' \rangle \in \delta_{\mathcal{P}}$, with *stmt* of the form $\alpha(p_1, \ldots, p_n); g; x_1, \ldots, x_k := e_1, \ldots, e_k$, such that $\sigma \models g[\bar{d}/\bar{p}]$, and for each $x_i \in X_{\mathcal{P}}$, we have

1. $\sigma'(x_i) = \sigma(e_i)$ when $e_i \in X_{\mathcal{P}}$, and
2. $\sigma'(x_i) = d_j$ when e_i is formal parameter p_j.

For a sequence of symbols $w = \alpha_1(\bar{d}_1) \cdots \alpha_m(\bar{d}_m)$, we write $\langle \ell, \sigma \rangle \stackrel{w}{\Longrightarrow}_{\mathcal{P}} \langle \ell', \sigma' \rangle$ to denote that there is a sequence of steps $\langle \ell, \sigma \rangle \xrightarrow{\alpha_1(\bar{d}_1)}_{\mathcal{P}} \langle \ell_1, \sigma_1 \rangle \cdots \langle \ell_{m-1}, \sigma_{m-1} \rangle \xrightarrow{\alpha_m(\bar{d}_m)}_{\mathcal{P}} \langle \ell', \sigma' \rangle$. We write $\langle \ell, \sigma \rangle \Longrightarrow_{\mathcal{P}} \langle \ell', \sigma' \rangle$ to express that there exists a w such that $\langle \ell, \sigma \rangle \stackrel{w}{\Longrightarrow}_{\mathcal{P}} \langle \ell', \sigma' \rangle$. A state $\langle \ell, \sigma \rangle$ is *reachable* if $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle \stackrel{w}{\Longrightarrow}_{\mathcal{P}} \langle \ell, \sigma \rangle$ for some w. A *trace* of $\mathcal{P}$ is a sequence w such that $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle \stackrel{w}{\Longrightarrow}_{\mathcal{P}} \langle \ell, \sigma \rangle$ for some $\langle \ell, \sigma \rangle$. The trace is *quiescent* if $\sigma \models \mathit{Quie}_{\mathcal{P}}(\ell)$. We let $T_{\mathcal{P}}$ denote the set of traces of $\mathcal{P}$, and let $Q_{\mathcal{P}}$ denote the set of quiescent traces of $\mathcal{P}$.

A symbol $\alpha(\bar{d})$ is *enabled* in a state $\langle \ell, \sigma \rangle$ if there is a step $\langle \ell, \sigma \rangle \xrightarrow{\alpha(\bar{d})}_{\mathcal{P}} \langle \ell', \sigma' \rangle$ for some state $\langle \ell', \sigma' \rangle$. Intuitively, $\sigma \models g^{\alpha}_{\ell}[\bar{d}/\bar{p}]$ iff $\alpha(\bar{d})$ is enabled in $\langle \ell, \sigma \rangle$.

Computing pre- and postconditions The algorithms in later sections use, as a basic building block, the computation of pre- and postconditions of statements. Let us first consider postconditions. Let φ be a formula and let g be a guard. Let $\overline{x}'$ be a vector of the same length as $\overline{x}$, containing fresh variables. The postcondition with respect to an assignment and a guard, respectively, are computed in the standard way as

$$\mathit{post}(\overline{x} := \overline{e}\ ;\ \varphi) := \exists \overline{x}'.\ (\varphi[\overline{x}'/\overline{x}] \wedge \overline{x} = \overline{e}[\overline{x}'/\overline{x}]) \quad \text{and} \quad \mathit{post}(g\ ;\ \varphi) := \varphi \wedge g.$$

Putting the above rules together, we derive the postcondition of a statement as follows.

$$\mathit{post}(\alpha(\overline{p}); g; \overline{x} := \overline{e};\ \varphi) \ := \ \exists \overline{p}.\ \exists \overline{x}'.\ \left((g \wedge \varphi)[\overline{x}'/\overline{x}] \wedge \overline{x} = \overline{e}[\overline{x}'/\overline{x}]\right).$$

Let us next consider preconditions in the analogous way. The precondition of an assignment and a guard, respectively, is defined as

$$\mathit{pre}(\overline{x} := \overline{e}; \varphi) := \varphi[\overline{e}/\overline{x}] \quad \text{and} \quad \mathit{pre}(g; \varphi) := g \wedge \varphi,$$

and a precondition of a whole statement is obtained by putting these together:

$$pre(\alpha(\overline{p}); g; \overline{x} := \overline{e} \ ; \ \varphi) \ = \ \exists \overline{p}. \ (g \wedge \varphi[\overline{e}/\overline{x}]) \ \ .$$

The existential quantifiers that arise in both the post- and precondition computation can, in the equality domain, be eliminated by a procedure involving two steps, *saturation* and *elimination*. Assume that φ is a formula in disjunctive normal form (DNF) over equalities and negated equalities over a set of variables. Formula $\exists(x_1, \ldots, x_n). \ \varphi$ is obtained by transforming every clause of φ in the following way:

1. *Saturation:* compute the reflexive, transitive and symmetric closure of the equality relation $=$ as partially given by the respective clause. This induces a partition on the set of variables. Let $[x]_=$ denote the class of this partition containing x. Then, transform φ by replacing every conjunct $x = y$ with $\bigwedge_{x' \in [x]_=, y' \in [y]_=} x' = y'$, and $x \neq y$ with $\bigwedge_{x' \in [x]_=, y' \in [y]_=} x' \neq y'$.
2. *Elimination:* from the saturated predicate, remove all conjuncts involving some of $x_1, \ldots, x_n$. Thus, the conjunct $x = y$ ($x \neq y$) is removed (or replaced with *true*) if x or y (or both) are in $\{x_1, \ldots, x_n\}$.

Example. Let φ be the predicate $x = y \wedge y \neq z$. For eliminating the existential quantifier in $\exists(y, z). \ \varphi$, we first saturate φ, which yields $x = y \wedge y \neq z \wedge x \neq z$. Since every conjunct in the saturated formula involves either y or z (or both), elimination leaves us with the empty conjunction, which is equivalent to *true*.

3 Refinement

We adapt the refinement relation between interface automata [7] to our processes with data, and at the same time we extend the definition to include quiescence.

Definition 2. Let $\mathcal{S}$ and $\mathcal{P}$ be processes. We say that $\mathcal{S}$ is *refined* by $\mathcal{P}$, denoted $\mathcal{P} \sqsubseteq \mathcal{S}$, if $I_\mathcal{S} \subseteq I_\mathcal{P}$, $O_\mathcal{P} \subseteq O_\mathcal{S}$, and whenever t is a trace of both $\mathcal{S}$ and $\mathcal{P}$ (i.e., $t \in T_\mathcal{S} \cap T_\mathcal{P}$), then

1. for any input symbol i, if $ti \in T_\mathcal{S}$ then $ti \in T_\mathcal{P}$,
2. for any output symbol o, if $to \in T_\mathcal{P}$ then $to \in T_\mathcal{S}$,
3. if t is quiescent in $\mathcal{P}$, then it is also quiescent in $\mathcal{S}$ (i.e., $Q_\mathcal{P} \cap T_\mathcal{S} \subseteq Q_\mathcal{S}$). $\square$

Condition 1 reflects the assumption that the environment does not supply unenabled input symbols. This assumption must not be strengthened by refinement; hence $\mathcal{P}$ must be prepared to accept any input that $\mathcal{S}$ can accept. Condition 2 reflects that the set of enabled output symbols constrains what the process may produce, which must not be weakened by refinement: hence $\mathcal{P}$ may at most produce the outputs that may be produced by $\mathcal{S}$. Condition 3 similarly reflects the view that allowed quiescence is viewed as a constraint on a process.

Given two processes, refinement can be checked by first computing the set of pairs of states of $\mathcal{S}$ and $\mathcal{P}$ which can be reached by the same trace, and thereafter checking

Algorithm 1: Computing $Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ for all $\ell_{\mathcal{S}} \in L_{\mathcal{S}}$ and $\ell_{\mathcal{P}} \in L_{\mathcal{P}}$

```
1 Initialise $Reach$ as $\varphi_0$ for $\langle \ell^0_{\mathcal{S}}, \ell^0_{\mathcal{P}} \rangle$ and false otherwise;
2 repeat
3     $changed := false$;
4     forall the $\langle \ell_{\mathcal{S}}, stmt_{\mathcal{S}}, \ell'_{\mathcal{S}} \rangle \in \delta_{\mathcal{S}}$ and $\langle \ell_{\mathcal{P}}, stmt_{\mathcal{P}}, \ell'_{\mathcal{P}} \rangle \in \delta_{\mathcal{P}}$ with equal actions do
5         $\phi := jointpost(stmt_{\mathcal{S}}, stmt_{\mathcal{P}}; Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle))$;
6         if $IsSat(\phi \wedge \neg Reach(\langle \ell'_{\mathcal{S}}, \ell'_{\mathcal{P}} \rangle))$ then
7             $Reach(\langle \ell'_{\mathcal{S}}, \ell'_{\mathcal{P}} \rangle) := Reach(\langle \ell'_{\mathcal{S}}, \ell'_{\mathcal{P}} \rangle) \vee \phi$;
8             $changed := true$;
9 until $\neg changed$;
```

the conditions in Definition 2 on these states. As the state space is infinite due to the unboundedness of $\mathcal{D}$, the set of reachable states has to be computed symbolically.

Let us assume (without loss of generality) that $X_{\mathcal{S}} \cap X_{\mathcal{P}} = \emptyset$. To check whether $\mathcal{P} \sqsubseteq \mathcal{S}$, we compute for each pair $\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle$ of locations a predicate $Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ over $X_{\mathcal{S}} \cup X_{\mathcal{P}}$ such that for an $X_{\mathcal{S}}$-valuation $\sigma_{\mathcal{S}}$ and $X_{\mathcal{P}}$-valuation $\sigma_{\mathcal{P}}$, we have $\sigma_{\mathcal{S}} \cup \sigma_{\mathcal{P}} \models Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ iff there is a trace $w \in T_{\mathcal{S}} \cap T_{\mathcal{P}}$ such that $\langle \ell^0_{\mathcal{S}}, \sigma^0_{\mathcal{S}} \rangle \overset{w}{\Longrightarrow}_{\mathcal{S}} \langle \ell_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ and $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle \overset{w}{\Longrightarrow}_{\mathcal{P}} \langle \ell_{\mathcal{P}}, \sigma_{\mathcal{P}} \rangle$.

We compute values of $Reach$ by a repeated postcondition computation, starting from the strongest condition that holds for the pair of initial states of $\mathcal{S}$ and $\mathcal{P}$. Let φ_0 be the strongest predicate satisfied by the union of the initial valuations $\sigma^0_{\mathcal{S}} \cup \sigma^0_{\mathcal{P}}$ of $\mathcal{S}$ and $\mathcal{P}$. This predicate expresses precisely that all variables have their initial values. The computation of $Reach$ is then carried out by a standard fixpoint procedure, shown in Algorithm 1.

It is initialized by letting $Reach(\langle \ell^0_{\mathcal{S}}, \ell^0_{\mathcal{P}} \rangle)$ be φ_0, and letting any other $Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ be *false*. Therafter the predicates $Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ are iteratively extended: for each pair of transitions $\langle \ell_{\mathcal{S}}, stmt_{\mathcal{S}}, \ell'_{\mathcal{S}} \rangle$ and $\langle \ell_{\mathcal{P}}, stmt_{\mathcal{P}}, \ell'_{\mathcal{P}} \rangle$ with the same action, we calculate the postcondition ϕ of the joint transition, in which $\mathcal{S}$ and $\mathcal{P}$ synchronize on a common symbol $\alpha(\overline{d})$. More precisely, letting $stmt_{\mathcal{S}}$ be $\alpha(\overline{p}); g_{\mathcal{S}}; \overline{x}_{\mathcal{S}} := \overline{e}_{\mathcal{S}}$, letting $stmt_{\mathcal{P}}$ be $\alpha(\overline{p}); g_{\mathcal{P}}; \overline{x}_{\mathcal{P}} := \overline{e}_{\mathcal{P}}$, and letting φ be a formula over $\overline{x}_{\mathcal{S}}$ and $\overline{x}_{\mathcal{P}}$, we define the *joint postcondition* of $stmt_{\mathcal{S}}$ and $stmt_{\mathcal{P}}$ wrp. to φ, denoted $jointpost(stmt_{\mathcal{S}}, stmt_{\mathcal{P}};\ \varphi)$ as

$$jointpost(stmt_{\mathcal{S}}, stmt_{\mathcal{P}};\ \varphi) \;=\; \exists \overline{p}.\ \exists \overline{x}'_{\mathcal{S}}, \overline{x}'_{\mathcal{P}}.\ \begin{bmatrix} \quad (g_{\mathcal{S}} \wedge g_{\mathcal{P}} \wedge \varphi)[\overline{x}'_{\mathcal{S}}, \overline{x}'_{\mathcal{P}} / \overline{x}_{\mathcal{S}}, \overline{x}_{\mathcal{P}}] \\ \wedge\ \overline{x}_{\mathcal{S}} = \overline{e}_{\mathcal{S}}[\overline{x}'_{\mathcal{S}} / \overline{x}_{\mathcal{S}}] \\ \wedge\ \overline{x}_{\mathcal{P}} = \overline{e}_{\mathcal{P}}[\overline{x}'_{\mathcal{P}} / \overline{x}_{\mathcal{P}}] \end{bmatrix}$$

where $\overline{x}'_{\mathcal{S}}$ is a tuple containing fresh variables, of the same length as $\overline{x}_{\mathcal{S}}$, and similarly for $\overline{x}'_{\mathcal{P}}$. In the algorithm, the predicates $Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$ are iteratively extended: for each pair of transitions $\langle \ell_{\mathcal{S}}, stmt_{\mathcal{S}}, \ell'_{\mathcal{S}} \rangle$ and $\langle \ell_{\mathcal{P}}, stmt_{\mathcal{P}}, \ell'_{\mathcal{P}} \rangle$ with the same action, we calculate their joint postcondition in $\phi := jointpost(stmt_{\mathcal{S}}, stmt_{\mathcal{P}}; Reach(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle))$. As long as $Reach(\langle \ell'_{\mathcal{S}}, \ell'_{\mathcal{P}} \rangle)$ is not weaker than ϕ, it is weakened by updating it to the disjunction of its previous value and ϕ. This process is repeated until convergence.

The following proposition states that, after Reach has been computed, checking refinement $\mathcal{P} \sqsubseteq \mathcal{S}$ amounts to checking validity of three conditions for each pair $\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle$ of locations.

Proposition 1. *$\mathcal{P} \sqsubseteq \mathcal{S}$ if and only if $I_{\mathcal{S}} \subseteq I_{\mathcal{P}}$, $O_{\mathcal{P}} \subseteq O_{\mathcal{S}}$, and for each pair $\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle$ of locations of $\mathcal{S}$ and $\mathcal{P}$ it holds that*

1. *$\mathit{Reach}(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle) \Rightarrow (g^{\alpha}_{\ell_{\mathcal{S}}} \Rightarrow g^{\alpha}_{\ell_{\mathcal{P}}})$ for all input actions $\alpha \in I_{\mathcal{S}}$,*
2. *$\mathit{Reach}(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle) \Rightarrow (g^{\alpha}_{\ell_{\mathcal{P}}} \Rightarrow g^{\alpha}_{\ell_{\mathcal{S}}})$ for all output actions $\alpha \in O_{\mathcal{P}}$, and*
3. *$\mathit{Reach}(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle) \Rightarrow (\mathit{Quie}_{\mathcal{P}}(\ell_{\mathcal{P}}) \Rightarrow \mathit{Quie}_{\mathcal{S}}(\ell_{\mathcal{S}}))$.* $\square$

Proof. First, assume that $\mathcal{P} \sqsubseteq \mathcal{S}$. We exemplify the proof idea focusing on the first condition only. Let $\sigma_{\mathcal{S}} \cup \sigma_{\mathcal{P}} \models \mathit{Reach}(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$, i.e., there exists a trace $w \in T_{\mathcal{S}} \cap T_{\mathcal{P}}$ such that $\langle \ell^0_{\mathcal{S}}, \sigma^0_{\mathcal{S}} \rangle \xRightarrow{w}_{\mathcal{S}} \langle \ell_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ and $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle \xRightarrow{w}_{\mathcal{P}} \langle \ell_{\mathcal{P}}, \sigma_{\mathcal{P}} \rangle$. Furthermore, let $i = \alpha(\overline{d})$, $\alpha \in I_{\mathcal{S}}$, be any input symbol such that $\sigma_{\mathcal{S}} \models g^{\alpha}_{\ell_{\mathcal{S}}}[\overline{d}/\overline{p}]$, i.e., $\alpha(\overline{d})$ is enabled in $\langle \ell_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ and hence $wi \in T_{\mathcal{S}}$. Since $\mathcal{P} \sqsubseteq \mathcal{S}$ implies that $wi \in T_{\mathcal{P}}$, $\alpha(\overline{d})$ is enabled in $\langle \ell_{\mathcal{P}}, \sigma_{\mathcal{P}} \rangle$ as well, thus $\sigma_{\mathcal{P}} \models g^{\alpha}_{\ell_{\mathcal{P}}}[\overline{d}/\overline{p}]$, rendering the implication $g^{\alpha}_{\ell_{\mathcal{S}}} \Rightarrow g^{\alpha}_{\ell_{\mathcal{P}}}$ true.

For the converse, assume that $\mathcal{P} \not\sqsubseteq \mathcal{S}$. Then, at least one of the conditions of Definition 2 is violated. Again, we exemplify the proof idea by looking at the third condition only, i.e., assuming that there exists a trace $w \in T_{\mathcal{S}} \cap T_{\mathcal{P}}$ such that w is quiescent in $\mathcal{P}$ but not in $\mathcal{S}$. Let $\langle \ell^0_{\mathcal{S}}, \sigma^0_{\mathcal{S}} \rangle \xRightarrow{w}_{\mathcal{S}} \langle \ell_{\mathcal{S}}, \sigma_{\mathcal{S}} \rangle$ and $\langle \ell^0_{\mathcal{P}}, \sigma^0_{\mathcal{P}} \rangle \xRightarrow{w}_{\mathcal{P}} \langle \ell_{\mathcal{P}}, \sigma_{\mathcal{P}} \rangle$ (thus, $\sigma_{\mathcal{S}} \cup \sigma_{\mathcal{P}} \models \mathit{Reach}(\langle \ell_{\mathcal{S}}, \ell_{\mathcal{P}} \rangle)$). Since w is quiescent in $\mathcal{P}$, we have $\sigma_{\mathcal{P}} \models \mathit{Quie}_{\mathcal{P}}(\ell_{\mathcal{P}})$. Conversely, since w is not quiescent in $\mathcal{S}$, we have $\sigma_{\mathcal{S}} \not\models \mathit{Quie}_{\mathcal{S}}(\ell_{\mathcal{S}})$, and thus $\sigma_{\mathcal{S}} \cup \sigma_{\mathcal{P}} \not\models (\mathit{Quie}_{\mathcal{P}}(\ell_{\mathcal{P}}) \Rightarrow \mathit{Quie}_{\mathcal{S}}(\ell_{\mathcal{S}}))$. $\square$

4 Parallel Composition

In this section, we generalize parallel composition of interface automata [7] to our processes with data and quiescence. Intuitively, the parallel composition operator yields the combined effect of its operands running asynchronously, but synchronizing on common actions. We use a *broadcast* model of communication in which an output from a component can be received by multiple components. An input $?a(\overline{d})$ and output $!a(\overline{d})$ combine to form an output $!a(\overline{d})$. Here, the attributes ? and ! on actions (as in $!a(\overline{d})$) are not part of the actions, they serve only to remind that the action in question is an input or output in the considered context.

Product Operation Before defining parallel composition of processes, we will as an auxiliary building block define the *product* of two processes as the process obtained by letting them run in parallel, while synchronizing on common actions and ignoring communication mismatches.

Let us define the parallel composition of two statements $\mathit{stmt}_1 = \alpha(\overline{p}); g_1; \overline{x}_1 := \overline{e}_1$ and $\mathit{stmt}_2 = \alpha(\overline{p}); g_2; \overline{x}_2 := \overline{e}_2$ with the same action α,[4] in two processes with disjoint sets of variables, as $\mathit{stmt}_1 \| \mathit{stmt}_2 = \alpha(\overline{p}); g_1 \wedge g_2; \overline{x}_1, \overline{x}_2 := \overline{e}_1, \overline{e}_2$.

We can now define product of two processes.

[4] Similarly to many programming languages, we assume that actions only have positional arguments, i.e., their formal parameters are identified solely by their order of occurrence in $\overline{p}$, not their name (if any).

Definition 3. Let $\mathcal{P}$ and $\mathcal{Q}$ be two processes. Then define $\delta_{\mathcal{P}\otimes\mathcal{Q}}$ as the set of transitions between product locations in $L_{\mathcal{P}} \times L_{\mathcal{Q}}$, which is obtained from $\delta_{\mathcal{P}}$ and $\delta_{\mathcal{Q}}$ as follows.

- If $\langle \ell_{\mathcal{P}}, \mathit{stmt}, \ell'_{\mathcal{P}} \rangle \in \delta_{\mathcal{P}}$ has an action which is not an action of $\mathcal{Q}$, (i.e., it is non-synchronizing), then $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle, \mathit{stmt}, \langle \ell'_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle \rangle \in \delta_{\mathcal{P}\otimes\mathcal{Q}}$ for any location $\ell_{\mathcal{Q}} \in L_{\mathcal{Q}}$.
- Symmetrically, if $\langle \ell_{\mathcal{Q}}, \mathit{stmt}, \ell'_{\mathcal{Q}} \rangle \in \delta_{\mathcal{Q}}$ has an action which is not an action of $\mathcal{P}$, then $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle, \mathit{stmt}, \langle \ell_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle \rangle \in \delta_{\mathcal{P}\otimes\mathcal{Q}}$ for any location $\ell_{\mathcal{P}} \in L_{\mathcal{P}}$.
- If $\langle \ell_{\mathcal{P}}, \mathit{stmt}_{\mathcal{P}}, \ell'_{\mathcal{P}} \rangle \in \delta_{\mathcal{P}}$ and $\langle \ell_{\mathcal{Q}}, \mathit{stmt}_{\mathcal{Q}}, \ell'_{\mathcal{Q}} \rangle \in \delta_{\mathcal{Q}}$ have the same action, i.e., they synchronize, then $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle, \mathit{stmt}_{\mathcal{P}} \| \mathit{stmt}_{\mathcal{Q}}, \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle \rangle \in \delta_{\mathcal{P}\otimes\mathcal{Q}}$. $\square$

Note that while we restrict ourselves to defining a product transition relation $\delta_{\mathcal{P}\otimes\mathcal{Q}}$ instead of a complete product operation $\mathcal{P} \otimes \mathcal{Q}$ (mostly because there is no reasonable choice for $Quie_{\mathcal{P}\otimes\mathcal{Q}}$), we nonetheless adapt the notation $\overset{w}{\Longrightarrow}_{\mathcal{P}\otimes\mathcal{Q}}$ for sequences of steps, as established in Section 2.

Parallel Composition We can now define parallel composition of processes. Note that there does not always exist a parallel composition of two processes. A necessary (but not sufficient, cf. Definition 5) precondition that has to be met is stated in the following definition.

Definition 4. *Two processes $\mathcal{P}$ and $\mathcal{Q}$ are* composable *if $O_{\mathcal{P}} \cap O_{\mathcal{Q}} = \emptyset$.*[5]

Intuitively, the parallel composition will be obtained from the product by restricting input transitions so that the product cannot reach an *illegal* state. A state of the product is illegal if one of the processes can generate an output symbol in their joint set of symbols which the other cannot receive. We say that a state of the product is *unsafe* if the product can reach an illegal state by a sequence of output steps. The parallel composition is now obtained by restricting input transitions so that they do not reach unsafe states.

Let us now formalize this intuition. First, we define a mapping, denoted $\mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}$ from pairs of locations of $\mathcal{P}$ and $\mathcal{Q}$ to the formula

$$\mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle) := \bigvee_{\alpha \in O_{\mathcal{P}} \cap I_{\mathcal{Q}}} \exists \overline{p}.\ (g^{\alpha}_{\ell_{\mathcal{P}}} \wedge \neg g^{\alpha}_{\ell_{\mathcal{Q}}}) \ \vee \bigvee_{\alpha \in O_{\mathcal{Q}} \cap I_{\mathcal{P}}} \exists \overline{p}.\ (g^{\alpha}_{\ell_{\mathcal{Q}}} \wedge \neg g^{\alpha}_{\ell_{\mathcal{P}}})$$

Intuitively, $\mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle)$ is true if $\mathcal{P}$ in location $\ell_{\mathcal{P}}$ can produce a synchronizing output symbol for which $\mathcal{Q}$ does not have a matching input step, or vice versa. Thereafter, we perform the pruning process, by defining the mapping $\mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}$ from pairs of locations of $\mathcal{P}$ and $\mathcal{Q}$ to formulas over $X_{\mathcal{P}} \cup X_{\mathcal{Q}}$ such that $\sigma_{\mathcal{P}} \cup \sigma_{\mathcal{Q}} \models \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle)$ iff there exists a sequence of symbols $w \in (O_{\mathcal{P}} \cup O_{\mathcal{Q}})^*$ such that $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle, \sigma_{\mathcal{P}} \cup \sigma_{\mathcal{Q}} \rangle \overset{w}{\Longrightarrow}_{\mathcal{P}\otimes\mathcal{Q}} \langle \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle, \sigma'_{\mathcal{P}} \cup \sigma'_{\mathcal{Q}} \rangle$ and $\sigma'_{\mathcal{P}} \cup \sigma'_{\mathcal{Q}} \models \mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle)$. The mapping $\mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}$ can be computed in a fashion similar to the computation of Reach, as illustrated in Algorithm 2.

[5] Formally, composing $\mathcal{P}$ and $\mathcal{Q}$ also requires $X_{\mathcal{P}}$ and $X_{\mathcal{Q}}$ to be disjoint. However, this can be assumed without loss of generality (as remarked in Section 3), as renaming variables does not change the behaviour of a process.

Algorithm 2: Computing $\mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}$ for product of $\mathcal{P}$ and $\mathcal{Q}$

```
1 $\mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}} := \mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}$;
2 repeat
3     $\mathit{changed} := \mathit{false}$;
4     forall the output transitions $\langle \ell, \mathit{stmt}, \ell' \rangle$ of $\delta_{\mathcal{P}\otimes\mathcal{Q}}$ do
5         $\phi := \mathit{pre}(\mathit{stmt};\ \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\ell'))$;
6         if $\mathit{IsSat}(\phi \wedge \neg \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\ell))$ then
7             $\mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\ell) := \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\ell) \vee \phi$;
8             $\mathit{changed} := \mathit{true}$;
9 until $\neg \mathit{changed}$;
```

Definition 5. Let $\mathcal{P}$ and $\mathcal{Q}$ be processes. The parallel composition of $\mathcal{P}$ and $\mathcal{Q}$ exists if and only if 1. $\mathcal{P}$ and $\mathcal{Q}$ are composable, and 2. $\sigma^0_{\mathcal{P}} \cup \sigma^0_{\mathcal{Q}} \not\models \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell^0_{\mathcal{P}}, \ell^0_{\mathcal{Q}} \rangle)$. In this case, it is the process $\mathcal{P}\|\mathcal{Q}$, obtained as
$\mathcal{P}\|\mathcal{Q} = (I_{\mathcal{P}\|\mathcal{Q}}, O_{\mathcal{P}\|\mathcal{Q}}, L_{\mathcal{P}\|\mathcal{Q}}, \ell^0_{\mathcal{P}\|\mathcal{Q}}, X_{\mathcal{P}\|\mathcal{Q}}, \sigma^0_{\mathcal{P}\|\mathcal{Q}}, \delta_{\mathcal{P}\|\mathcal{Q}}, \mathit{Quie}_{\mathcal{P}\|\mathcal{Q}})$, where

- $I_{\mathcal{P}\|\mathcal{Q}} = (I_{\mathcal{P}} \cup I_{\mathcal{Q}}) \setminus O_{\mathcal{P}\|\mathcal{Q}}$,
- $O_{\mathcal{P}\|\mathcal{Q}} = O_{\mathcal{P}} \cup O_{\mathcal{Q}}$,
- $L_{\mathcal{P}\|\mathcal{Q}} = L_{\mathcal{P}} \times L_{\mathcal{Q}}$,
- $\ell^0_{\mathcal{P}\|\mathcal{Q}} = \langle \ell^0_{\mathcal{P}}, \ell^0_{\mathcal{Q}} \rangle$,
- $X_{\mathcal{P}\|\mathcal{Q}} = X_{\mathcal{P}} \cup X_{\mathcal{Q}}$,
- $\sigma^0_{\mathcal{P}\|\mathcal{Q}} = \sigma^0_{\mathcal{P}} \cup \sigma^0_{\mathcal{Q}}$,
- $\delta_{\mathcal{P}\|\mathcal{Q}}$ is obtained from $\delta_{\mathcal{P}\otimes\mathcal{Q}}$ by strengthening every guard g of every input transitions of form $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle,\ \alpha(\overline{p}); g; \overline{x} := \overline{e},\ \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle \rangle$ in $\delta_{\mathcal{P}\otimes\mathcal{Q}}$ to
$$g \wedge \mathit{pre}(\overline{x} := \overline{e}; \neg \mathit{Unsafe}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{Q}} \rangle))\ \ ,$$
- $\mathit{Quie}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle) = \mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle) \vee (\mathit{Quie}_{\mathcal{P}}(\ell_{\mathcal{P}}) \wedge \mathit{Quie}_{\mathcal{Q}}(\ell_{\mathcal{Q}}))$. □

An important observation is that parallel composition preserves the consistency requirement introduced in Section 2.

Proposition 2. *Let $\mathcal{P}$ and $\mathcal{Q}$ be processes such that the parallel composition $\mathcal{P} \parallel \mathcal{Q}$ exists, and that $\mathcal{P}$ and $\mathcal{Q}$ are consistent with $\mathit{Quie}_{\mathcal{P}}$ and $\mathit{Quie}_{\mathcal{Q}}$, respectively. Then, $\mathcal{P} \parallel \mathcal{Q}$ is consistent with $\mathit{Quie}_{\mathcal{P}\|\mathcal{Q}}$, i.e., for any location $\ell_{\mathcal{P}\|\mathcal{Q}} \in L_{\mathcal{P}\|\mathcal{Q}}$,*

$$\mathit{Quie}_{\mathcal{P}\|\mathcal{Q}}(\ell_{\mathcal{P}\|\mathcal{Q}}) \vee \bigvee_{\alpha \in O_{\mathcal{P}\|\mathcal{Q}}} \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{P}\|\mathcal{Q}}}$$

is universally valid.

Proof. Let $\ell_{\mathcal{P}\|\mathcal{Q}} = \langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle$, and let $\sigma_{\mathcal{P}\|\mathcal{Q}}$ be a valuation such that $\sigma_{\mathcal{P}\|\mathcal{Q}} \not\models \mathit{Quie}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle)$, i.e., we have $\sigma_{\mathcal{P}\|\mathcal{Q}} \not\models \mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle)$ and either of $\sigma_{\mathcal{P}\|\mathcal{Q}} \not\models \mathit{Quie}_{\mathcal{P}}(\ell_{\mathcal{P}})$ or $\sigma_{\mathcal{P}\|\mathcal{Q}} \not\models \mathit{Quie}_{\mathcal{Q}}(\ell_{\mathcal{Q}})$ (or both). Without loss of generality, we only consider the first case. Due to the consistency of $\mathcal{P}$, there exists an output action $\alpha \in O_{\mathcal{P}}$ such that $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{P}}}$. If α is not an (input) action of $\mathcal{Q}$, then, by definition of the parallel composition, we have $g^{\alpha}_{\ell_{\mathcal{P}\|\mathcal{Q}}} \equiv g^{\alpha}_{\ell_{\mathcal{P}}}$ (note that only guards of input transitions are strengthened in Definition 5) and thus $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{P}\|\mathcal{Q}}}$. Otherwise, we have $g^{\alpha}_{\ell_{\mathcal{P}\|\mathcal{Q}}} \equiv g^{\alpha}_{\ell_{\mathcal{P}}} \wedge g^{\alpha}_{\ell_{\mathcal{Q}}}$. From $\sigma_{\mathcal{P}\|\mathcal{Q}} \not\models \mathit{Illegal}_{\mathcal{P}\|\mathcal{Q}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{Q}} \rangle)$ we can conclude that $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \forall \overline{p}.\ (g^{\alpha}_{\ell_{\mathcal{P}}} \Rightarrow g^{\alpha}_{\ell_{\mathcal{Q}}})$. Combining this with $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{P}}}$ yields $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{Q}}}$, and thus $\sigma_{\mathcal{P}\|\mathcal{Q}} \models \exists \overline{p}.\ g^{\alpha}_{\ell_{\mathcal{P}\|\mathcal{Q}}}$. □

The following proposition establishes the important property that refinement is preserved by parallel composition.

Proposition 3. *Let $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{S}$ and $\mathcal{T}$ be processes with $\mathcal{P} \sqsubseteq \mathcal{S}$ and $\mathcal{Q} \sqsubseteq \mathcal{T}$, such that the parallel composition $\mathcal{S} \parallel \mathcal{T}$ exists. Then the parallel composition $\mathcal{P} \parallel \mathcal{Q}$ exists and $\mathcal{P} \parallel \mathcal{Q} \sqsubseteq \mathcal{S} \parallel \mathcal{T}$.* □

5 Quotient

In this section, we introduce the *quotient* operation, which can be seen as an "inverse" of parallel composition. It is connected to the synthesis problem in the following way: given a specification for a system $\mathcal{R}$, together with a component $\mathcal{P}$ implementing part of $\mathcal{R}$, the quotient, denoted $\mathcal{R} \setminus \mathcal{P}$, yields the *least refined* (in the sense of $\sqsubseteq$) process for the remaining part of $\mathcal{R}$, i.e., such that $\mathcal{P} \parallel (\mathcal{R} \setminus \mathcal{P}) \sqsubseteq \mathcal{R}$. Therefore, quotient can be thought of as an adjoint of parallel composition.

Looking at the treatment of sets of actions in Definition 5, we see that a necessary requirement for the existence of a quotient is that $O_{\mathcal{P}} \subseteq O_{\mathcal{R}}$. Then, the set of output actions of the quotient must be $O_{\mathcal{R}} \setminus O_{\mathcal{P}}$. However, there is some freedom for the set $I_{\mathcal{R}\setminus\mathcal{P}}$ of input actions of the quotient. From Definition 5, we take as a natural choice $I_{\mathcal{R}\setminus\mathcal{P}}$ to be $(O_{\mathcal{P}} \cup I_{\mathcal{R}})$, since a quotient with these actions will always exist if there is one with a smaller set, and since the subsequently presented technique to produce a quotient will not have to consider the difficulties that come with actions that are not visible to the quotient.

Computing the Quotient Let us provide an algorithmic construction of the quotient. The structure of our construction is the following. We first construct the product of $\mathcal{P}$ and $\mathcal{R}$ using the product construction of Definition 3. We thereafter construct $\mathcal{R} \setminus \mathcal{P}$ so that it ensures that the combination of $\mathcal{P}$ and $\mathcal{R}$ does not reach an undesired state. An undesired state occurs if 1. $\mathcal{P}$ can produce an output which cannot be produced by $\mathcal{R}$, or 2. $\mathcal{R}$ can receive an input which cannot be received by $\mathcal{P}$, or 3. the state of $\mathcal{R}$ is not quiescent, but it is a *deadlock*, that is, no output transition of $\mathcal{R}$ can be taken without $\mathcal{R} \setminus \mathcal{P}$ losing control (i.e., $\mathcal{R} \setminus \mathcal{P}$ will no longer be able to ensure that an undesired state will not eventually be reached).

To enforce the above criteria, we define a mapping Bad from pairs of locations of $\mathcal{P}$ and $\mathcal{R}$ to predicates over $X_{\mathcal{P}} \cup X_{\mathcal{R}}$ such that $\sigma \models \mathit{Bad}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle)$ iff there exists a sequence of symbols $w \in O^{*}_{\mathcal{P}}$ such that $\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle, \sigma \rangle \xRightarrow{w}_{\mathcal{P}\otimes\mathcal{R}} \langle \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{R}} \rangle, \sigma' \rangle$ and $\langle \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{R}} \rangle, \sigma' \rangle$ is an undesired state according to any (or all) of the above criteria. In the following, we describe how Bad can be computed algorithmically. As a first step, we define a mapping $\mathit{NotRefine}_{\mathcal{R}\setminus\mathcal{P}}$ from pairs of locations of $\mathcal{P}$ and $\mathcal{R}$ to predicates such

Algorithm 3: Computing Bad

```
1  $Bad := NotRefine_{\mathcal{R}\setminus\mathcal{P}}$;
2  repeat
3      $changed := false$;
4      forall the $O_{\mathcal{P}}$-transitions $\langle \ell, stmt, \ell' \rangle$ of $\delta_{\mathcal{P}\otimes\mathcal{R}}$ do
5          $\phi := pre(stmt; Bad(\ell'))$;
6          if $IsSat(\phi \wedge \neg Bad(\ell))$ then
7              $Bad(\ell) := Bad(\ell) \vee \phi$;
8              $changed := true$;
9      forall the $\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle \in L_{\mathcal{P}} \times L_{\mathcal{R}}$ do
10         $\phi := Deadlock^{Bad}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle) \wedge \neg Quie_{\mathcal{R}}(\ell_{\mathcal{R}})$;
11         if $IsSat(\phi \wedge \neg Bad(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle))$ then
12             $Bad(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle) := Bad(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle) \vee \phi$;
13             $changed := true$;
14 until $\neg changed$;
```

that $NotRefine_{\mathcal{R}\setminus\mathcal{P}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle)$ is true in the situations that should be avoided according to Criterion 1 or 2 above. We can represent $NotRefine_{\mathcal{R}\setminus\mathcal{P}}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle)$ as the formula

$$\bigvee_{\alpha \in I_{\mathcal{R}} \cap I_{\mathcal{P}}} (g^{\alpha}_{\ell_{\mathcal{R}}} \wedge \neg g^{\alpha}_{\ell_{\mathcal{P}}}) \;\vee\; \bigvee_{\alpha \in O_{\mathcal{P}}} (g^{\alpha}_{\ell_{\mathcal{P}}} \wedge \neg g^{\alpha}_{\ell_{\mathcal{R}}})$$

To formalize Criterion 3, we first define deadlock as a predicate parameterized by the set of states which are uncontrollable (represented by the predicate Bad), i.e., from which $\mathcal{R} \setminus \mathcal{P}$ cannot guarantee avoiding undesired states. We define $Deadlock^{Bad}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle)$ as

$$\neg \bigvee\nolimits_{\langle \langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle, stmt, \langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{R}} \rangle \rangle \in \delta^{O_{\mathcal{R}}}_{\mathcal{P}\otimes\mathcal{R}}} pre(stmt;\ \neg Bad(\langle \ell'_{\mathcal{P}}, \ell'_{\mathcal{R}} \rangle))$$

where $\delta^{O_{\mathcal{R}}}_{\mathcal{P}\otimes\mathcal{R}}$ are the $O_{\mathcal{R}}$-transitions of $\delta_{\mathcal{P}\otimes\mathcal{R}}$. Notice that $Deadlock^{Bad}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle)$ is automatically *true* if $\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle$ has no output transitions. A state $\langle \langle \ell_{\mathcal{R}}, \ell_{\mathcal{P}} \rangle, \sigma \rangle$ is undesired according to Criterion 3 above if

$$\sigma \;\models\; Deadlock^{Bad}(\langle \ell_{\mathcal{P}}, \ell_{\mathcal{R}} \rangle) \;\wedge\; \neg Quie_{\mathcal{R}}(\ell_{\mathcal{R}})\ .$$

The complete computation of Bad is illustrated in Algorithm 3.

After Bad is computed, we compute $\mathcal{R} \setminus \mathcal{P}$ as the process $\mathcal{R} \setminus \mathcal{P} = (I_{\mathcal{R}\setminus\mathcal{P}}, O_{\mathcal{R}\setminus\mathcal{P}}, L_{\mathcal{R}\setminus\mathcal{P}}, \ell^0_{\mathcal{R}\setminus\mathcal{P}}, X_{\mathcal{R}\setminus\mathcal{P}}, \sigma^0_{\mathcal{R}\setminus\mathcal{P}}, \delta_{\mathcal{R}\setminus\mathcal{P}}, Quie_{\mathcal{R}\setminus\mathcal{P}})$ where

– $I_{\mathcal{R}\setminus\mathcal{P}} = O_{\mathcal{P}} \cup I_{\mathcal{R}}$,
– $O_{\mathcal{R}\setminus\mathcal{P}} = O_{\mathcal{R}} \setminus O_{\mathcal{P}}$,
– $L_{\mathcal{R}\setminus\mathcal{P}} = L_{\mathcal{P}} \times L_{\mathcal{R}}$,
– $\ell^0_{\mathcal{R}\setminus\mathcal{P}} = \langle \ell^0_{\mathcal{P}}, \ell^0_{\mathcal{R}} \rangle$,
– $X_{\mathcal{R}\setminus\mathcal{P}} = X_{\mathcal{P}} \cup X_{\mathcal{R}}$,
– $\sigma^0_{\mathcal{R}\setminus\mathcal{P}} = \sigma^0_{\mathcal{P}} \cup \sigma^0_{\mathcal{Q}}$,

– $\delta_{\mathcal{R}\setminus\mathcal{P}}$ is obtained from $\delta_{\mathcal{P}\otimes\mathcal{R}}$ by strengthening every guard g of every $O_{\mathcal{R}\setminus\mathcal{P}}$-transition of the form $\langle\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}}\rangle,\ \alpha(\overline{p});g;\overline{x}:=\overline{e},\ \langle\ell'_{\mathcal{P}},\ell'_{\mathcal{R}}\rangle\rangle$ in $\delta_{\mathcal{P}\otimes\mathcal{R}}$ to

$$g \wedge pre(\overline{x}:=\overline{e};\neg Bad(\langle\ell'_{\mathcal{P}},\ell'_{\mathcal{R}}\rangle))$$

– $Quie_{\mathcal{R}\setminus\mathcal{P}} = Quie_{\mathcal{P}}(\ell_{\mathcal{P}}) \Rightarrow Quie_{\mathcal{R}}(\ell_{\mathcal{R}})$

The input transitions of $\mathcal{R}\setminus\mathcal{P}$ are the same as the input $I_{\mathcal{R}\setminus\mathcal{P}}$-transitions of $\delta_{\mathcal{P}\otimes\mathcal{R}}$. Output transitions of $\mathcal{R}\setminus\mathcal{P}$ will be obtained from $O_{\mathcal{R}\setminus\mathcal{P}}$-transitions of $\delta_{\mathcal{P}\otimes\mathcal{R}}$ by equipping them with additional guards which keep computations of $(\mathcal{R}\setminus\mathcal{P})\parallel\mathcal{P}$ outside Bad and hence also outside undesired states.

The following proposition states that the quotient, if it exists, is indeed the most general component that can cooperate with $\mathcal{P}$ to satisfy $\mathcal{R}$.

Proposition 4. *Let $\mathcal{P}$ and $\mathcal{R}$ be such that $O_{\mathcal{P}}\subseteq O_{\mathcal{R}}$ and $I_{\mathcal{P}}\subseteq(I_{\mathcal{R}}\cup O_{\mathcal{R}})$. If $\mathcal{R}\setminus\mathcal{P}$ as computed in this section exists, then*

– *$\mathcal{P}\parallel(\mathcal{R}\setminus\mathcal{P})\sqsubseteq R$, and*
– *for any $\mathcal{Q}$ with $O_{\mathcal{Q}}=O_{\mathcal{R}\setminus\mathcal{P}}$ and $I_{\mathcal{Q}}=I_{\mathcal{R}\setminus\mathcal{P}}$ such that $\mathcal{P}\parallel\mathcal{Q}\sqsubseteq R$, we have $\mathcal{Q}\sqsubseteq(\mathcal{R}\setminus\mathcal{P})$.* $\square$

Pruning the Quotient The quotient obtained by the above method may contain a significant amount of redundancy. Particularly, 1. some of its states may never be reached, 2. some transitions may never be taken, and 3. some parts of conditions within guards may be true for every computation reaching the source location of the transition.

To obtain a more compact solution, we will prune the redundant parts of $\mathcal{R}\setminus\mathcal{P}$. Using a procedure analogous to Algorithm 1 in Section 3, we compute for every location $\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}\setminus\mathcal{P}}\rangle$ of $\mathcal{P}\parallel(\mathcal{R}\setminus\mathcal{P})$ a predicate $Reach(\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}\setminus\mathcal{P}}\rangle)$ such that a state $\langle\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}\setminus\mathcal{P}}\rangle,\sigma\rangle$ is reachable in $\mathcal{P}\parallel(\mathcal{R}\setminus\mathcal{P})$ iff σ satisfies $Reach(\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}\setminus\mathcal{P}}\rangle)$. As before, we assume (w.l.o.g.) that $X_{\mathcal{R}\setminus\mathcal{P}}\cap X_{\mathcal{P}}=\emptyset$; further, let us assume that $X_{\mathcal{P}}$ is the tuple $x_1^{\mathcal{P}},\ldots,x_k^{\mathcal{P}}$. Then, for every location $\ell_{\mathcal{R}\setminus\mathcal{P}}$ of $\mathcal{R}\setminus\mathcal{P}$, we will compute

$$Reach(\ell_{\mathcal{R}\setminus\mathcal{P}}) = \exists x_1^{\mathcal{P}},\ldots,x_k^{\mathcal{P}}.\ \bigvee_{\ell_{\mathcal{P}}\in L_{\mathcal{P}}} Reach(\langle\ell_{\mathcal{P}},\ell_{\mathcal{R}\setminus\mathcal{P}}\rangle)$$

which characterizes all possible valuations of variables of $\mathcal{R}\setminus\mathcal{P}$ that can appear in a computation of $\mathcal{P}\parallel(\mathcal{R}\setminus\mathcal{P})$ together with $\ell_{\mathcal{R}\setminus\mathcal{P}}$. We then prune the redundancies of the types 1–3 above from $\mathcal{R}\setminus\mathcal{P}$ as follows:

1. Remove locations $\ell_{\mathcal{R}\setminus\mathcal{P}}$ where $IsSat(Reach(\ell_{\mathcal{R}\setminus\mathcal{P}}))$ is *false*.
2. Remove transitions $\langle\ell_{\mathcal{R}\setminus\mathcal{P}},\ \alpha(\overline{p});g;\overline{x}:=\overline{e},\ \ell'_{\mathcal{R}\setminus\mathcal{P}}\rangle$ where $IsSat(Reach(\ell_{\mathcal{R}\setminus\mathcal{P}})\wedge g)$ is *false*.
3. For each remaining transition $\langle\ell_{\mathcal{R}\setminus\mathcal{P}},\ \alpha(\overline{p});g;\overline{x}:=\overline{e},\ \ell'_{\mathcal{R}\setminus\mathcal{P}}\rangle$, we weaken the guard g to $Reach(\ell_{\mathcal{R}\setminus\mathcal{P}})\Rightarrow g$, possibly enabling further simplification of the formula (e.g., by transforming it to DNF and removing redundant literals and clauses).

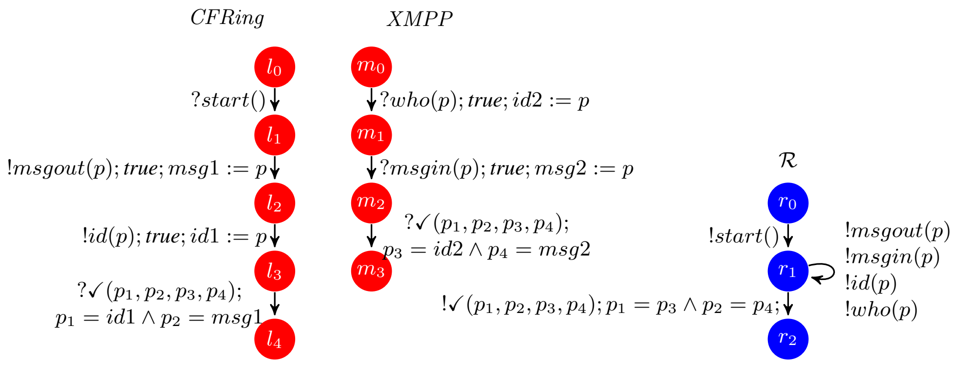

CFRing
XMPP
$\mathcal{R}$
l_0
$?start()$
l_1
$!msgout(p); \mathit{true}; msg1 := p$
l_2
$!id(p); \mathit{true}; id1 := p$
l_3
$?\checkmark(p_1, p_2, p_3, p_4);$
$p_1 = id1 \wedge p_2 = msg1$
l_4
m_0
$?who(p); \mathit{true}; id2 := p$
m_1
$?msgin(p); \mathit{true}; msg2 := p$
m_2
$?\checkmark(p_1, p_2, p_3, p_4);$
$p_3 = id2 \wedge p_4 = msg2$
m_3
r_0
$!start()$
r_1
$!msgout(p)$
$!msgin(p)$
$!id(p)$
$!who(p)$
$!\checkmark(p_1, p_2, p_3, p_4); p_1 = p_3 \wedge p_2 = p_4;$
r_2

Fig. 1. Messaging clients CFRing and XMPP, on the left. The synthesis problem is specified by the process $\mathcal{R}$ on the right.

6 Applications to the Synthesis of Mediators

We demonstrate our framework and our implementation on examples from mediator synthesis. A *mediator* is a process that mediates communication between several parties with incompatible interfaces, ensuring that they interact to achieve a certain aim, while preventing any communication mismatches. We demonstrate how this task can be specified as a problem of computing a quotient for a specification $\mathcal{R}$ which can be automatically generated from this aim. We have implemented the computation of quotient, and illustrate its application on mediator synthesis in e-commerce applications.

Messaging Protocol In the first example, the scenario consists of two incompatible messaging protocols, CFRing and XMPP, both of which need to communicate with one another through a mediator that we must construct. The two messaging clients are represented by the two processes shown on the left in Figure 1. In the description, we omit guards of transitions that are *true*. Variables are initialized by a special value $\perp$.

- The process CFRing has input actions $start$, output actions $msgout$ and id, variables $msg1$ and $id1$, and locations $\{l_0, l_1, l_2, l_3, l_4\}$.
- The process XMPP has input actions $msgin$ and who, variables $msg2$ and $id2$, and locations $\{m_0, m_1, m_2, m_3\}$.

The goal of the communication is that the two clients agree on the message and the id. That is, the values of variables $msg1$ and $id1$ of $CFRing$ should on termination be equal to the values of $msg2$ and $id2$ of $XMPP$. This goal can be captured by adding a special action $\checkmark$, which is performed at the end, and which reveals the values of the relevant variables of CFRing and XMPP. The specification $\mathcal{R}$ allows the components to perform any sequence of actions, but requires that quiescence is reached only after jointly performing the $\checkmark$ action, where the revealed variables satisfy the desired goal constraints. In other words, the communication parties are forced to reveal the values of their variables within action $\checkmark$, and $\mathcal{R}$ checks that they correspond. $\mathcal{R}$ does not specify what exactly happens before, it only states that the trace should begin with $start$ and may continue by any sequence of actions $msgout$, $msgin$, id, and who. To ensure that

$$\begin{tikzcd}
\langle l_0, m_0\rangle \arrow[r, "?start()"] \arrow[d, "?who(p); id2 := p"'] & \langle l_1, m_0\rangle \arrow[r, "\begin{array}{c}!msgout(p); \mathit{true};\\ msg1 := p\end{array}"] \arrow[d] & \langle l_2, m_0\rangle \arrow[r, "\begin{array}{c}!id(p); \mathit{true};\\ id1 := p\end{array}"] \arrow[d] & \langle l_3, m_0\rangle \arrow[d] & \mathcal{P} \\
\langle l_0, m_1\rangle \arrow[r] \arrow[d, "?msgin(p); msg2 := p"'] & \langle l_1, m_1\rangle \arrow[r] \arrow[d] & \langle l_2, m_1\rangle \arrow[r] \arrow[d] & \langle l_3, m_1\rangle \arrow[d] & \\
\langle l_0, m_2\rangle \arrow[r] & \langle l_1, m_2\rangle \arrow[r] & \langle l_2, m_2\rangle \arrow[r] & \langle l_3, m_2\rangle \arrow[dr, "\begin{array}{c}?\checkmark(p_1, p_2, p_3, p_4);\\ p_1 = id1 \wedge p_2 = msg1 \wedge p_3 = id2 \wedge p_4 = msg2\end{array}"'] & \\
 & & & & \langle l_4, m_3\rangle
\end{tikzcd}$$

$$\begin{tikzcd}
\langle\langle l_0, m_0\rangle, r_0\rangle \arrow[r, "start()"] & \langle\langle l_1, m_0\rangle, r_1\rangle \arrow[r, "\begin{array}{c}msgout(p); \mathit{true};\\ msg1 := p\end{array}"] \arrow[d, "who(p); \mathit{true}; id2 := p"'] & \langle\langle l_2, m_0\rangle, r_1\rangle \arrow[r, "\begin{array}{c}id(p); \mathit{true};\\ id1 := p\end{array}"] \arrow[d] & \langle\langle l_3, m_0\rangle, r_1\rangle \arrow[d] & \delta_{\mathcal{P}\otimes\mathcal{R}} \\
 & \langle\langle l_1, m_1\rangle, r_1\rangle \arrow[r] \arrow[d, "msgin(p); \mathit{true}; msg2 := p"'] & \langle\langle l_2, m_1\rangle, r_1\rangle \arrow[r] \arrow[d] & \langle\langle l_3, m_1\rangle, r_1\rangle \arrow[d] & \\
 & \langle\langle l_1, m_2\rangle, r_1\rangle \arrow[r] & \langle\langle l_2, m_2\rangle, r_1\rangle \arrow[r] & \langle\langle l_3, m_2\rangle, r_1\rangle \arrow[dr, "\begin{array}{c}\checkmark(p_1, p_2, p_3, p_4);\\ p_1 = id1 \wedge p_2 = msg1 \wedge p_3 = id2 \wedge p_4 = msg2 \wedge\\ p_1 = p_3 \wedge p_2 = p_4\end{array}"'] & \\
 & & & & \langle\langle l_4, m_3\rangle, r_2\rangle
\end{tikzcd}$$

Fig. 2. The process $\mathcal{P} = \mathit{CFRing} \parallel \mathit{XMPP}$ and the transition relation $\delta_{\mathcal{P}\otimes\mathcal{R}}$. The statements of the transitions marked by vertical arrows are defined by the labels on the left margin of the figure. The statements of the transitions marked by horizontal arrows are defined by the labels on the top margin of the figure.

$\checkmark$ will indeed be performed eventually, that is, that both processes reach their terminal states, $\mathit{Quie}_{\mathcal{R}}$ is defined to be *false* for r_0, r_1 and *true* for r_2.

We follow the scheme of the previous sections, and compute first the process $\mathcal{P} = \mathit{CFRing} \parallel \mathit{XMPP}$ shown in the upper part of Figure 2, which inherits all the transitions of CFRing and XMPP unsynchronized. We are looking for the least refined process such that its composition with $\mathcal{P}$ refines $\mathcal{R}$, that is, we are looking for the quotient $\mathcal{R} \setminus \mathcal{P}$. The next step is the construction of the transition relation $\delta_{\mathcal{P}\otimes\mathcal{R}}$ which is shown in Figure 2 (lower part). We proceed by computing the mapping Bad that characterizes uncontrollable states of $\mathcal{R} \setminus \mathcal{P}$ (from where $\mathcal{R} \setminus \mathcal{P}$ cannot guarantee that no undesired state is reached). This is done by running Algorithm 3.

Let us discuss some steps of the computation of Bad in detail. The undesired states here are only those which deadlock and are not quiescent according to $\mathcal{R}$, that is, the location of $\mathcal{R}$ is different from r_2. Since $\mathit{Bad}(\ell)$ is initially *false* for all locations of $\mathcal{R} \setminus \mathcal{P}$, the computation starts by evaluating $\mathit{Deadlock}^{\mathit{Bad}}$. Let us pick the location $\ell = \langle\langle l_3, m_2\rangle, r_1\rangle$. $\mathit{Deadlock}^{\mathit{Bad}}(\ell)$ evaluates initially to the disjunction $\varphi \equiv id1 \neq id2 \vee msg1 \neq msg2$, which falsifies the guard of the only output transition of ℓ (leading to $\langle\langle l_4, m_3\rangle, r_2\rangle$). We set $\mathit{Bad}(\ell)$ to φ and we then propagate $\mathit{Bad}(\ell) = \varphi$ backwards via output transitions of $\mathcal{R}$. The precondition of φ with respect to the transition from $\ell' = \langle\langle l_2, m_2\rangle, r_1\rangle$ to ℓ is evaluated as *true* (particularly, the disjunct $id2 \neq id1$ is first transformed into $id2 = p$ which is then turned into *true* by

$$\begin{tikzcd}
\langle\langle l_0, m_0\rangle, r_0\rangle \arrow[r, "!start()"] & \langle\langle l_1, m_0\rangle, r_1\rangle \arrow[r, "{\begin{array}{c} ?msgout(p); \mathit{true}; \\ msg1' := p \end{array}}"] & \langle\langle l_2, m_0\rangle, r_1\rangle \arrow[r, "{\begin{array}{c} ?id(p); \mathit{true}; \\ id1' := p \end{array}}"] & \langle\langle l_3, m_0\rangle, r_1\rangle \arrow[d, "{!who(p); p = id1'; id2' := p}"'] & \mathcal{R} \setminus \mathcal{P} \\
 & & & \langle\langle l_3, m_1\rangle, r_1\rangle \arrow[d, "{!msgin(p); p = msg1'; msg2' := p}"'] & \\
 & & & \langle\langle l_3, m_2\rangle, r_1\rangle \arrow[dr, "{!\checkmark(id1', msg1', id2', msg2')}"'] & \\
 & & & & \langle\langle l_4, m_3\rangle, r_2\rangle
\end{tikzcd}$$

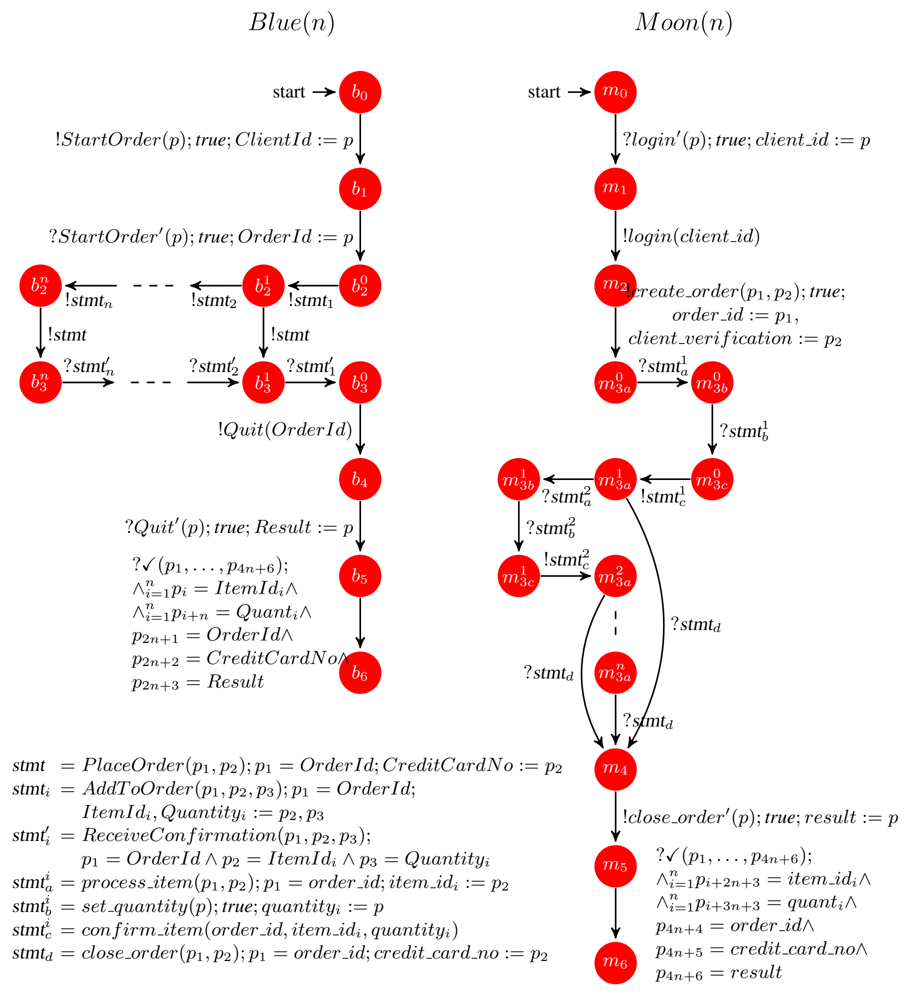
Fig. 3. The resulting mediator $\mathcal{R} \setminus \mathcal{P}$ for the application with messaging.

the existential quantification over p). Hence, we set $Bad(\ell')$ to *true*; therefore, $\mathcal{R} \setminus \mathcal{P}$ must avoid visiting ℓ' in all situations. Note that it is indeed the case that if $\mathcal{R} \setminus \mathcal{P}$ allows $XMPP$ to reach state m_2 (where the value of $msg2$ is already fixed) before $msg1$ is generated by $CFRing$, then $\mathcal{R} \setminus \mathcal{P}$ cannot guarantee $msg1 = msg2$ at the end of the communication which may lead to an undesired deadlock. Let us now look at the precondition of φ wrt. the transition from $\langle\langle l_3, m_1\rangle, r_1\rangle$. Since ℓ is its successor and we have set $Bad(\ell) = \varphi$, $Deadlock^{Bad}(\langle\langle l_3, m_1\rangle, r_1\rangle)$ evaluates to $id2 \neq id1$ (particularly, the disjunct $msg1 \neq msg2$ turns into $msg1 \neq p$ which after the universal quantification over p becomes *false*). (Particularly, $\neg Bad(\ell) \equiv id1 = id2 \wedge msg1 = msg2$, $pre(\neg\varphi, msgin(p); msg2 := p)$ evaluates to $id1 = id2$ which is then negated.) Intuitively, this reflects the fact that when $CFRing$ is in state l_3 and $XMPP$ in state m_1, the values of $id1$ and $id2$ should be already equal since they cannot become equal otherwise. The rest of the symbolic backward computation of Bad is carried out analogously.

The construction of $\mathcal{R} \setminus \mathcal{P}$ continues by adding guards to the transitions of $\delta_{\mathcal{P}\otimes\mathcal{R}}$ to guarantee that any computation stays outside of Bad. Finally, we prune unreachable states, useless transitions, and redundant guard conditions, as described in Section 5. The resulting quotient $\mathcal{R} \setminus \mathcal{P}$ is shown in Figure 3.

E-Commerce The next example is a more realistic and larger system. Due to its size, we only explain the specification and the functionality of the system and the synthesized mediator. We are given both a client and a customer service application, shown in Figure 4, that together are supposed to realize an e-commerce workflow, but are incompatible. The client $Blue$ starts by sending a $StartOrder$ message containing its id and expects to receive an id of a new order. It then orders a number of items in some quantities using the $AddToOrder$ action, provides its payment information via the $PlaceOrder$ action, and expects all items together with their quantities to be confirmed by the customer service. It blocks in case that it does not receive the right confirmation. $Blue$ then announces that it is ready to quit the transaction and expects to receive the result of the payment transaction, $Result$ (indicating whether or not the payment transaction was successful).

The customer service $Moon$ expects to receive the client's id, then it sends a confirmation and sends an id of a new order, together with a client verification. It is then prepared to repeat a loop in which it 1) receives an order of an item, 2) receives a quan-


$Blue(n)$
$Moon(n)$
start
b_0
$!StartOrder(p); true; ClientId := p$
b_1
$?StartOrder'(p); true; OrderId := p$
b_2^0
$!stmt_1$
b_2^1
$!stmt_2$
$!stmt_n$
b_2^n
$!stmt$
$!stmt$
b_3^n
$?stmt'_n$
$?stmt'_2$
b_3^1
$?stmt'_1$
b_3^0
$!Quit(OrderId)$
b_4
$?Quit'(p); true; Result := p$
b_5
$?\checkmark(p_1, \ldots, p_{4n+6});$
$\wedge_{i=1}^{n} p_i = ItemId_i \wedge$
$\wedge_{i=1}^{n} p_{i+n} = Quant_i \wedge$
$p_{2n+1} = OrderId \wedge$
$p_{2n+2} = CreditCardNo \wedge$
$p_{2n+3} = Result$
b_6
start
m_0
$?login'(p); true; client\_id := p$
m_1
$!login(client\_id)$
m_2
$!create\_order(p_1, p_2); true;$
$order\_id := p_1,$
$client\_verification := p_2$
m_{3a}^0
$?stmt_a^1$
m_{3b}^0
$?stmt_b^1$
m_{3c}^0
$!stmt_c^1$
m_{3a}^1
$?stmt_a^2$
m_{3b}^1
$?stmt_b^2$
m_{3c}^1
$!stmt_c^2$
m_{3a}^2
$?stmt_d$
m_{3a}^n
$?stmt_d$
$?stmt_d$
m_4
$!close\_order'(p); true; result := p$
m_5
$?\checkmark(p_1, \ldots, p_{4n+6});$
$\wedge_{i=1}^{n} p_{i+2n+3} = item\_id_i \wedge$
$\wedge_{i=1}^{n} p_{i+3n+3} = quant_i \wedge$
$p_{4n+4} = order\_id \wedge$
$p_{4n+5} = credit\_card\_no \wedge$
$p_{4n+6} = result$
m_6

$$
\begin{aligned}
stmt &= PlaceOrder(p_1, p_2); p_1 = OrderId; CreditCardNo := p_2\\
stmt_i &= AddToOrder(p_1, p_2, p_3); p_1 = OrderId;\\
&\quad ItemId_i, Quantity_i := p_2, p_3\\
stmt'_i &= ReceiveConfirmation(p_1, p_2, p_3);\\
&\quad p_1 = OrderId \wedge p_2 = ItemId_i \wedge p_3 = Quantity_i\\
stmt_a^i &= process\_item(p_1, p_2); p_1 = order\_id; item\_id_i := p_2\\
stmt_b^i &= set\_quantity(p); true; quantity_i := p\\
stmt_c^i &= confirm\_item(order\_id, item\_id_i, quantity_i)\\
stmt_d &= close\_order(p_1, p_2); p_1 = order\_id; credit\_card\_no := p_2
\end{aligned}
$$

Fig. 4. Client *Blue* and service *Moon*, parameterized by maximum number of ordered items n.

.

tity in which the item is ordered, and 3) confirms that the item in the given quantity is ordered. After that, it receives payment information, arranges the payment via a third-party service (which is invisible to the client and not modelled here), and sends the result of the payment transaction.

Ideally, we would like to model the scenario where the client can order any number of items. However, our modelling mechanism allows only processes with a finite number of variables. This scenario would require processes with an unbounded number of variables, both for the $Blue$ and $Moon$ services and for the specification, as well as for mediator. We therefore restrict ourselves to the case where the number of ordered items is bounded by a constant n, which becomes a parameter of the synthesis problem.

The specification (parameterized by the maximum number of ordered items n) is given by the process $\mathcal{R}(n)$ in Figure 5. $Quie_{\mathcal{R}}$ is defined as *true* for r_2 and *false* otherwise. Similar to the previous examples, $\mathcal{R}$ specifies that both sides finish the transaction and that at the end of the transaction, both sides agree on all the important values.

A mediator for a fixed number of ordered items n can be synthesized analogously to the previous example. We construct the process $\mathcal{P} = Blue(n) \parallel Moon(n)$ and synthesize the mediator in the form of the quotient $\mathcal{R} \setminus \mathcal{P}$ by: 1. computing the predicate Bad characterizing uncontrollable states of the product of $\mathcal{P}$ and $\mathcal{R}$, using Algorithm 3, 2. strengthening guards in $\delta_{\mathcal{P}\otimes\mathcal{R}}$ so that no uncontrollable state (and hence no undesirable state) can be reached, and 3. pruning the useless states and transitions, utilizing Algorithm 1. The synthesized mediator for the case $n = 2$ is shown in Figure 6. For simplicity, the figure displays only the $\mathcal{P}$-component of locations of $\mathcal{R} \setminus \mathcal{P}$, i.e., the locations of $Blue(2) \parallel Moon(2)$. The location $\langle b_0, m_0 \rangle$ is coupled with r_0, the location $\langle b_6, m_6 \rangle$ with r_2, and all the other displayed locations with r_1.

The functionality of the synthesized mediator can be explained as follows. It first brings the system to the point when $Blue$ has ordered its first item (the state $\langle b_2^1, m_{3a}^0 \rangle$). It can now decide to start forwarding the first order to $Moon$ (vertical transitions). At the same time, it has to be ready to receive either the second order or the credit card details from $Blue$ (the horizontal transitions). If $Blue$ ordered also the second item, the mediator will forward the second order to $Moon$ after it has forwarded the first one. The mediator will send the payment credit card details to $Moon$ only after it has received it from $Blue$ and after it has forwarded all orders of $Blue$ (this is taken care of by the guard of the statement stmt_d^*). It waits for $Blue$ to confirm all orders. Sending confirmations to $Blue$ is independent from receiving confirmations from $Moon$ since to send the right confirmations, the mediator only needs to know what was ordered by $Blue$. The mediator can therefore choose from many variants of interleaving the communication with $Blue$ and $Moon$.

7 Summary and Future Work

We have extended the theory of interface automata [7] with data and progress, using mechanisms that have been naturally adapted from similar other works in the literature. The resulting theory allows to capture data-flow behaviour at the modelling level, and can also be used to formulate the mediator synthesis problem. Further extensions can be done along several dimensions. One is to extend the theory to richer sets of relations and functions over the data domain. Another dimension is to handle non-deterministic

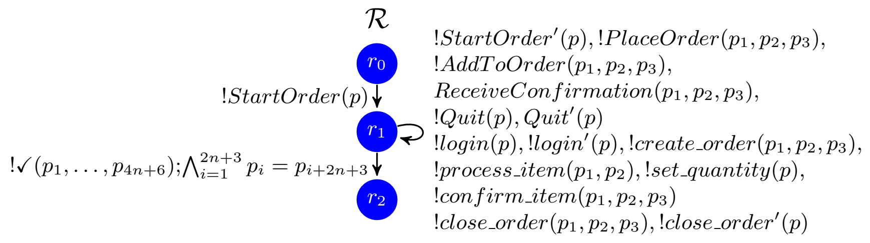

$\mathcal{R}$
r_0
$!StartOrder(p)$
r_1
$!\checkmark(p_1,\ldots,p_{4n+6});\bigwedge_{i=1}^{2n+3} p_i = p_{i+2n+3}$
r_2
$!StartOrder'(p), !PlaceOrder(p_1,p_2,p_3),$
$!AddToOrder(p_1,p_2,p_3),$
$ReceiveConfirmation(p_1,p_2,p_3),$
$!Quit(p), Quit'(p)$
$!login(p), !login'(p), !create\_order(p_1,p_2,p_3),$
$!process\_item(p_1,p_2), !set\_quantity(p),$
$!confirm\_item(p_1,p_2,p_3)$
$!close\_order(p_1,p_2,p_3), !close\_order'(p)$

Fig. 5. Specification of the synthesis problem where the number of ordered items is bounded by n is given by the process $\mathcal{R}(n)$.

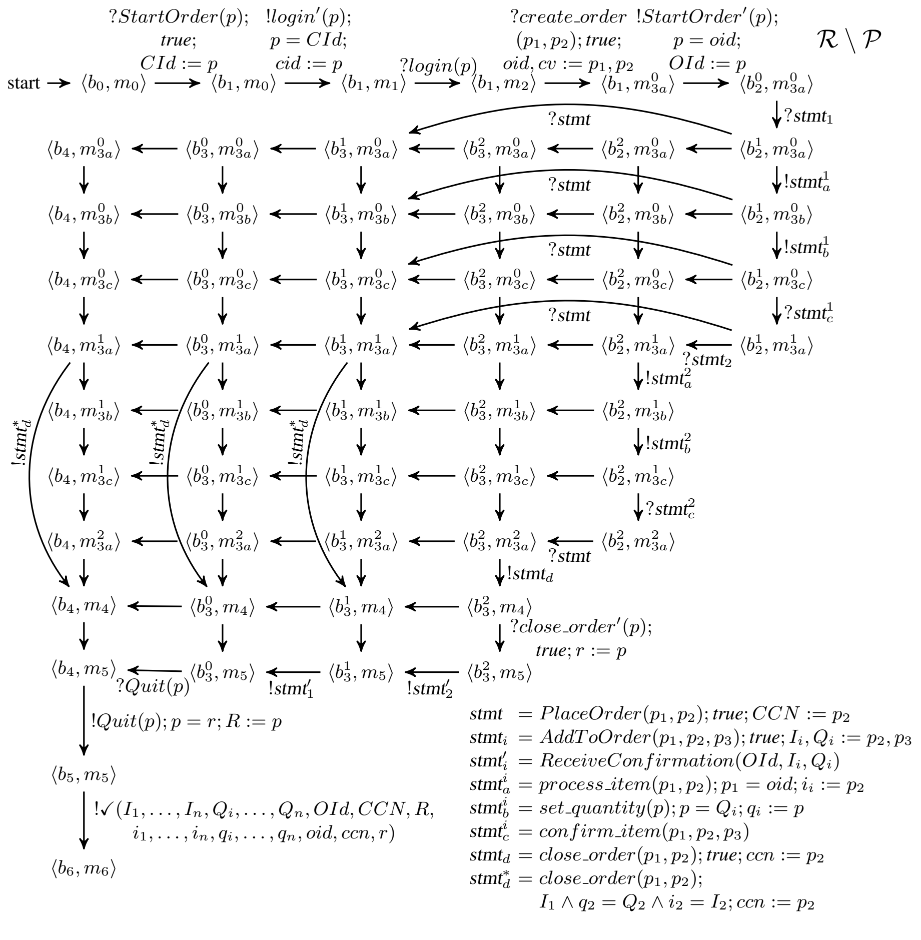

$\mathcal{R} \setminus \mathcal{P}$
start
$?StartOrder(p);$ $true;$ $CId := p$
$!login'(p);$ $p = CId;$ $cid := p$
$?login(p)$
$?create\_order$ $(p_1,p_2); true;$ $oid, cv := p_1, p_2$
$!StartOrder'(p);$ $p = oid;$ $OId := p$
$\langle b_0, m_0\rangle$
$\langle b_1, m_0\rangle$
$\langle b_1, m_1\rangle$
$\langle b_1, m_2\rangle$
$\langle b_1, m_{3a}^0\rangle$
$\langle b_2^0, m_{3a}^0\rangle$
$?stmt_1$
$?stmt$
$!stmt_a^1$
$!stmt_b^1$
$?stmt_c^1$
$?stmt_2$
$!stmt_a^2$
$!stmt_b^2$
$?stmt_c^2$
$!stmt_d$
$!stmt_d^*$
$?close\_order'(p);$ $true; r := p$
$?Quit(p)$
$!stmt_1'$
$!stmt_2'$
$!Quit(p); p = r; R := p$
$!\checkmark(I_1,\ldots,I_n,Q_i,\ldots,Q_n,OId,CCN,R,$ $i_1,\ldots,i_n,q_i,\ldots,q_n,oid,ccn,r)$
$\langle b_4, m_5\rangle$
$\langle b_5, m_5\rangle$
$\langle b_6, m_6\rangle$
$stmt = PlaceOrder(p_1,p_2); true; CCN := p_2$
$stmt_i = AddToOrder(p_1,p_2,p_3); true; I_i, Q_i := p_2, p_3$
$stmt_i' = ReceiveConfirmation(OId, I_i, Q_i)$
$stmt_a^i = process\_item(p_1,p_2); p_1 = oid; i_i := p_2$
$stmt_b^i = set\_quantity(p); p = Q_i; q_i := p$
$stmt_c^i = confirm\_item(p_1,p_2,p_3)$
$stmt_d = close\_order(p_1,p_2); true; ccn := p_2$
$stmt_d^* = close\_order(p_1,p_2);$ $I_1 \wedge q_2 = Q_2 \wedge i_2 = I_2; ccn := p_2$

Fig. 6. Resulting mediator $\mathcal{R} \setminus \mathcal{P}$ for the e-commerce service with $n = 2$. The labels of the vertical edges in a row agree with the label of the right-most edge in the row, the labels of the horizontal edges in a column agree with the label of the bottom edge of the column.

processes, which brings the problem of uncertainty – the mediator cannot be completely sure about the state of the systems it controls. A further dimension is to extend the specification framework to cover liveness properties. Such extensions have been considered for the finite-state case in the literature (see, e.g., [19]), but their adaption to handle data appears nontrivial.

Acknowledgements. This work is carried out as part of the European FP7 ICT FET CONNECT project (http://connect-forever.eu/). The last author was supported in part by the UPMARC centre of excellence.

References

1. S. Bauer, K. Larsen, A. Legay, U. Nyman, and A. Wasowski. A Modal Specification Theory for Components with Data. *Sci. Comput. Program.*, 83:106–128, 2014.
2. A. Bennaceur, C. Chilton, M. Isberner, and B. Jonsson. Automated Mediator Synthesis: Combining Behavioural and Ontological Reasoning. In *Proc. of SEFM*, volume 8137 of *LNCS*, pages 274–288. Springer, 2013.
3. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *Proc. of VLDB '05*. VLDB Endowment, 2005.
4. P. Bertoli, M. Pistore, and P. Traverso. Automated Composition of Web Services via Planning in Asynchronous Domains. *Artif. Intell.*, 174(3-4), 2010.
5. P. Bhaduri and S. Ramesh. Interface Synthesis and Protocol Conversion. *Form. Asp. Comput.*, 20(2):205–224, March 2008.
6. T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska. A Compositional Specification Theory for Component Behaviours. In *ESOP*, volume 7211 of *LNCS*, pages 148–168. Springer, 2012.
7. L. de Alfaro and T. A. Henzinger. Interface Automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.
8. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits*. PhD thesis, Carnegie Mellon University, 1988.
9. N. Guermouche, O. Perrin, and C. Ringeissen. A Mediator Based Approach For Services Composition. In *SERA '08*. IEEE, 2008.
10. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Behind the Curtain. In *Proc. 22nd ACM Symp. on Principles of Database Systems*, pages 1–14. ACM, 2003.
11. B. Jonsson. Compositional specification and verification of distributed systems. *ACM Trans. on Programming Languages and Systems*, 16(2):259–303, 1994.
12. K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *ESOP*, volume 4421 of *LNCS*, pages 64–79, 2007.
13. N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc.* 6^{th} *ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.
14. E. Olderog and C. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23(1):9–66, 1986.
15. J.-B. Raclet. Residual for component specifications. *Electronic Notes in Theoretical Computer Science*, 215:93–110, June 2008.
16. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. Modal Interfaces: Unifying Interface Automata and Modal Specifications. In *Proc. of EMSOFT '09*, pages 87–96. ACM, 2009.

17. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. A Modal Interface Theory for Component-based Design. *Fundamenta Informaticae*, 108(1-2):119–149, January 2011.
18. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are Modalities Good for Interface Theories? In *ACSD '09*, pages 119–127. IEEE, 2009.
19. J.-F. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger. Algorithms for Omega-Regular Games with Imperfect Information. *Logical Methods in CS*, 3(3), 2007.
20. L. Ryzhyk, P. Chubb, I. Kuz, E. L. Sueur, and G. Heiser. Automatic Device Driver Synthesis with Termite. In *SOSP'09*, pages 73–86. ACM, 2009.
21. J. Tretmans. Model-Based Testing and Some Steps Towards Test-Based Modelling. In *SFM*, volume 6659 of *LNCS*, pages 297–326. Springer, 2011.
22. D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.