

Component Substitutability via Equivalencies of Component-Interaction Automata

Ivana Černá, Pavlína Vařeková and Barbora Zimmerova^{1,2}

*Faculty of Informatics, Masaryk University
Brno, Czech Republic*

Abstract

We provide a new look at formal aspects of component substitutability (replacement of a component with a new one) and independent implementability (reuse of a component in any system where its implementation satisfies the specification given by the environment), in view of an underlying formalism called Component-interaction automata. Our aim is to offer a formal characterization of preconditions that lead to reconfiguration correctness (proper component substitution and safe independent implementation). Such preconditions then guarantee that the updated system remains equivalent to the former one and hence there is no need to verify it again.

The contribution of the paper is twofold. First, we formally define three relations that allows us to compare behaviours of two components with respect to reconfiguration correctness. Namely, the equivalence relation, specification–implementation relation, and substitutability relation. Second, we formally characterize the problem of component substitutability for both equivalent and non-equivalent components, and the problem of independent implementability. The characterizations are captured in several propositions which are proved in the text.

Keywords: component-based systems, reconfiguration correctness, component substitutability, independent implementability

1 Introduction

One of the essential benefits of component-based systems is their flexibility with respect to future changes. As the systems are composed of autonomous components, they may evolve simply by update of particular components. However, the components are often developed by third parties which brings new verification issues regarding correctness of interaction among such components. One of the issues, in view of component updates, is called reconfiguration correctness. The reconfiguration correctness comprises two specific problems, component substitutability (replacement of a component with a new one) and independent implementability (safe reuse of a component in any system where its implementation satisfies the specification given by its environment).

¹ Email: {cerna,xvareko1,zimmerova}@fi.muni.cz

² The authors have been supported by grants No. 1ET400300504 and GACR 201/06/1338

Our solution to these problems, as alternative to verification from scratch, is based on formal characterization of relationship between the new and the former component, which guarantees that the update will not break the existing functionality of an overall system. We regard the problem of independent implementability as the substitutability of component implementation for its specification which allows us to propose a uniform solution for both problems.

We rely on an underlying formalism called Component-interaction automata [15], which is briefly described in Sections 2 and 4. Section 3 introduces the notion of equivalence between two component-interaction automata, which is defined with respect to a given set of observable labels. This allows us to explicitly state the level of accuracy at which automata are compared, and hence study several kinds of equivalencies between automata. The characterization of reconfiguration correctness with respect to substitution of two equivalent components is proposed in Section 5. Sections 3 and 5 together provide a formal foundation stone for subsequent relations and results that are extensions of these. Sections 6 and 7 introduce the specification–implementation and the substitutability relations. In these sections we also characterize the problems of independent implementability and substitutability of non-equivalent components, and propose solutions based on the results of Section 5. Related work is discussed in Section 8 and we conclude in Section 9.

2 Component-interaction automata

The Component-interaction automata language [5,15] was designed for modelling of component interactions in hierarchical component-based software systems. It captures each component as a labelled transition system with structured labels (to remember components which communicated on an action) and a hierarchy of component names (which represents the architectural structure of the component). Such features allow the language to model component interactions in fine detail while the language is still generally usable for several variations of component-based systems (with different synchronization strategies for instance). The essential definitions are briefly reminded in this section.

A *hierarchy of component names* is a tuple $H = (H_1, \dots, H_n)$, $n \in \mathbb{N}$, of one of the following forms, S_H denotes the set of component names corresponding to H . The first case is that H_1, \dots, H_n are pairwise different natural numbers; then $S_H = \bigcup_{i=1}^n \{H_i\}$. The second case is that H_1, \dots, H_n are hierarchies of component names where S_{H_1}, \dots, S_{H_n} are pairwise disjoint; then $S_H = \bigcup_{i=1}^n S_{H_i}$.

A *component-interaction automaton* (or a *CI automaton* for short) is a 5-tuple $\mathcal{C} = (Q, Act, \delta, I, H)$ where Q is a finite set of states, Act is a finite set of *actions*, $\Sigma = ((S_H \cup \{-\}) \times Act \times (S_H \cup \{-\})) \setminus (\{-\} \times Act \times \{-\})$ is a set of *labels*, $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of *labelled transitions*, $I \subseteq Q$ is a nonempty set of *initial states*, and H is a hierarchy of component names. The labels have semantics of input, output, or internal, based on their structure, as indicated in Notation 2.1. Examples of two CI automata are in Figure 1.

A *path* of a CI automaton $\mathcal{C} = (Q, Act, \delta, I, H)$ is an alternating sequence of states and labels given by δ that is either infinite, or is finite in case that it ends with a state from which there is no transition in δ . The set of all paths of a CI

automaton \mathcal{C} is denoted $Path(\mathcal{C})$. The set of all finite prefixes of paths from $Path(\mathcal{C})$ that end with a state is denoted $FinPath(\mathcal{C})$.

Notation 2.1 For a given CI automaton $\mathcal{C} = (Q, Act, \delta, I, H)$ we denote

- $\mathcal{L}_{\mathcal{C}} = \{l \mid \exists q_0, l_0, \dots, q_{k-1}, l_{k-1}, q_k \in FinPath(\mathcal{C}) : q_0 \in I \wedge l_{k-1} = l\}$
the set of all labels reachable in \mathcal{C} ,
- $\mathcal{L}_{inp, \mathcal{C}} = \mathcal{L}_{\mathcal{C}} \cap \{(-, a, n_2) \mid a \in Act, n_2 \in \mathbb{N}\}$
the set of all input labels reachable in \mathcal{C} (a component n_2 inputs an action a),
- $\mathcal{L}_{out, \mathcal{C}} = \mathcal{L}_{\mathcal{C}} \cap \{(n_1, a, -) \mid a \in Act, n_1 \in \mathbb{N}\}$
the set of all output labels reachable in \mathcal{C} (a component n_1 outputs an action a),
- $\mathcal{L}_{int, \mathcal{C}} = \mathcal{L}_{\mathcal{C}} \cap \{(n_1, a, n_2) \mid a \in Act, n_1, n_2 \in \mathbb{N}\}$
the set of all internal labels reachable in \mathcal{C} (n_1 and n_2 synchronize on a),
- $\mathcal{L}_{ext, \mathcal{C}} = \mathcal{L}_{inp, \mathcal{C}} \cup \mathcal{L}_{out, \mathcal{C}} = \mathcal{L}_{\mathcal{C}} \setminus \mathcal{L}_{int, \mathcal{C}}$
the set of all external (input and output) labels reachable in \mathcal{C} .

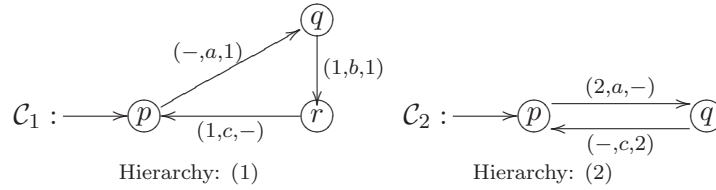


Fig. 1. Example of CI automata

3 Equivalence of component-interaction automata

This section introduces the equivalence of two CI automata defined as an equivalence with respect to observable steps X . The observable step of an automaton consists of a single observable transition (with a label from X) preceded and followed by an arbitrary number (potentially zero) of silent transitions (with labels outside X). Using this concept we define the equivalence of two CI automata in a similar way to Milner's weak bisimulation [13].

Definition 3.1 Let $\mathcal{C}' = (Q', Act', \delta', I', H')$ and $\mathcal{C}'' = (Q'', Act'', \delta'', I'', H'')$ be CI automata and X be a set of labels. A binary relation $\sim \subseteq Q' \times Q''$ is called an *observation equivalence of \mathcal{C}' and \mathcal{C}'' with respect to X* iff $q' \sim q''$ implies:

- (i) Whenever $(q', l, r') \in \delta'$ then $\exists q''l_0q_1l_1 \cdots q_nl_nr'' \in FinPath(\mathcal{C}'')$ satisfying $r' \sim r''$ and
 - if $l \notin X$ then $\{l_0, l_1, \dots, l_n\} \cap X = \emptyset$, (★)
 - if $l \in X$ then $\{l_0, l_1, \dots, l_n\} \cap X = \{l\} \wedge \exists! i \in \{0, 1, \dots, n\} : l_i = l$
where $\exists! i$ denotes that there is exactly one such index.
- (ii) Whenever $(q'', l, r'') \in \delta''$ then $\exists q'l_0q_1l_1 \cdots q_nl_nr' \in FinPath(\mathcal{C}')$ satisfying $r'' \sim r'$ and (★).

CI automata \mathcal{C}' , \mathcal{C}'' are equivalent with respect to X , $\mathcal{C}' \equiv_X \mathcal{C}''$, iff there is an observation equivalence \sim of \mathcal{C}' and \mathcal{C}'' with respect to X such that:

- For every $q' \in I'$ there is $q'' \in I''$ such that $q' \sim q''$. (★★)
- For every $q'' \in I''$ there is $q' \in I'$ such that $q' \sim q''$. (★★★)

The use of the parameter X in the definition of \equiv_X has a special significance. It allows us to explicitly state the level of accuracy at which the new and the former system are compared.

Remark 3.2 For CI automata \mathcal{C} and \mathcal{C}' some special cases of the set X are:

- (i) $X = \mathcal{L}_{\mathcal{C}} \cup \mathcal{L}_{\mathcal{C}'}$
An analogy of strong bisimulation where all labels are observable.
- (ii) $X = \mathcal{L}_{ext,\mathcal{C}} \cup \mathcal{L}_{ext,\mathcal{C}'}$
An analogy of weak bisimulation where all internal labels are silent.
- (iii) $X = \mathcal{L}_{\mathcal{C}} \cup \mathcal{L}_{ext,\mathcal{C}'}$
Refinement of \mathcal{C} by \mathcal{C}' where \mathcal{C}' must respect all transitions of \mathcal{C} (input, output, internal), but may perform other new internal transitions.
- (iv) $X = (\mathcal{L}_{\mathcal{C}} \cup \mathcal{L}_{\mathcal{C}'}) \setminus \{(n, a, n) \mid n \in \mathbb{N}\}$
Only inner internal labels of primitive components are silent.

Note that in all these cases, the set X includes all external labels of the automata, $X \supseteq \mathcal{L}_{ext,\mathcal{C}} \cup \mathcal{L}_{ext,\mathcal{C}'}$. This is natural as external labels may participate in communication with other components which influence an overall behaviour of the system.

It can be easily proved that, for any fixed set of labels X , the relation \equiv_X is an equivalence (reflexive, symmetric, and transitive) on the set of all CI automata. Moreover, it has a property, stated by the following lemma, of which we take advantage in following sections.

Lemma 3.3 *Let \mathcal{C} , \mathcal{C}' be CI automata, X be a set of labels. Then*

$$\text{if } \mathcal{C} \equiv_X \mathcal{C}' \text{ then } \forall X' \subseteq X : \mathcal{C} \equiv_{X'} \mathcal{C}'$$

Proof. As $\mathcal{C} \equiv_X \mathcal{C}'$, there exists an observation equivalence \sim of \mathcal{C} and \mathcal{C}' with respect to X satisfying $(\star\star)$ and $(\star\star\star)$. It can be easily proved that the same \sim is also an observation equivalence of \mathcal{C} and \mathcal{C}' with respect to any $X' \subseteq X$. \square

According to Definition 3.1, two CI automata may be equivalent only if their sets of reachable observable labels are identical. It among others means that they need to have the same names for functionally corresponding primitive components mentioned in observable labels. However, more practical issue is to check whether two automata behave the same no matter what the names of their primitive components are.

Notation 3.4 *Let S_H be a set of component names and X be a set of labels. By $S_{H,X}$ we denote the set of component names from S_H that appear in any label from X . Formally, $S_{H,X} = S_H \cap \{n \mid \exists a, x : (n, a, x) \in X \vee (x, a, n) \in X\}$.*

Definition 3.5 Let $\mathcal{C} = (Q, Act, \delta, I, H)$ be a CI automaton, let $M \subseteq \mathbb{N}$, and $r : M \rightarrow \mathbb{N}$ be a function, called *renaming function*. We say, that a CI automaton \mathcal{C}' results from \mathcal{C} after renaming of all component names with r , iff $\mathcal{C}' = (Q, Act, \delta', I, H')$ where δ' and H' results from δ , resp. H , by replacing every occurrence of any component name $x \in M$ with $r(x)$.

Definition 3.6 Let \mathcal{C} and \mathcal{C}' be CI automata, X be a set of labels. We say that a CI automaton \mathcal{C}' is *equivalent to \mathcal{C}* with respect to observable labels X *up to 1:1* (or *up*

to $1:N$) renaming, iff there is a bijection (resp. surjection) $r : S_{H',X} \rightarrow S_{H,X}$ such that the CI automaton \mathcal{C}'_r , which results from \mathcal{C}' after renaming of all component names with r , is equivalent to \mathcal{C} with respect to observable labels X , $\mathcal{C} \equiv_X \mathcal{C}'_r$.

Thanks to the information about participating components, which CI automata encompass in the labels, the equivalence may be assessed according to the correspondence of primitive components. If the renaming function is a bijection, for each observable component n_i in \mathcal{C} there must be *exactly one* observable component n_j in \mathcal{C}' performing an equivalent functionality. In case we do not want to make such a strict restriction, we may consider the case that for each observable component n_i in \mathcal{C} there may be a *set of components* in \mathcal{C}' forming a notional component n_j which performs an equivalent functionality. That is the second case when the renaming function is a surjection. Moreover, we could reason about another type of function that would join names of several components to a group to test whether we may replace them for another group of components with shifted responsibilities (some services moved from one component to another one within the group) even if we know that one by one replacement would not succeed. In addition, the concept of equivalence up to renaming can be applied also to other relations of CI automata, in particular those introduced in Sections 6 and 7.

4 Composition with respect to architectural assembly

We have presented the definition of CI automata and introduced the notion of equivalence between them. Now, before we proceed to component update within a system, one more issue needs to be discussed. It concerns the role of component's environment. As the component is interconnected with the rest of the system, the environment influence its behaviour coordinating the component. Hence the form of component interconnection with the rest of the system must be taken into consideration before the component substitution. This interconnection is in fact determined by the type of composition that was used to compose the system together.

Component-interaction automata offer a parameterizable composition operator which composes given automata in a way that it preserves only the transitions that are really feasible in a system. For example, let us consider a system consisting of three components C_1 , C_2 , and C_3 , where both C_1 and C_2 provide a service a and C_3 requires a . Imagine that only C_1 and C_3 are connected by communicational binding, C_2 and C_3 are not. Then the composition respects that only C_1 and C_3 may synchronize. The synchronization of C_2 and C_3 is syntactically possible, but not feasible in the system. One of the ways to provide such composition is to parameterize the composition operator with a set of labels \mathcal{F} which represents the bindings via which the components may communicate (in component assembly of the system). This is possible thanks to the structure of labels (in CI automata) which contain the names of components between which the actions are communicated. Such composition operator is denoted $\otimes^{\mathcal{F}}$ and its definition and several properties with respect to \equiv_X are presented in this section.

The composition of CI automata is defined as an operation that for a composable

indexed set³ of CI automata and an additional parameter \mathcal{F} returns the composite automaton. A set of CI automata $\mathcal{S} = \{(Q_i, Act_i, \delta_i, I_i, H_i)\}_{i \in \mathcal{I}}$ is composable if $\mathcal{I} \subseteq \mathbb{N}$ is finite and $(H_i)_{i \in \mathcal{I}}$ is a hierarchy of component names⁴. The composite CI automaton over \mathcal{S} is defined with help of a complete transition space over \mathcal{S} denoted $\Delta_{\mathcal{S}}$. The complete transition space consists of transitions between product states for given automata, such that from each state, there are just the transitions of single component automata, and the transitions caused by synchronization of two component automata on a complementary label.

Definition 4.1 Let \mathcal{F} be a set of labels, then $\otimes^{\mathcal{F}}$ denotes a unary composition operator on composable sets of CI automata. If $\mathcal{S} = \{(Q_i, Act_i, \delta_i, I_i, H_i)\}_{i \in \mathcal{I}}$ is a composable set of CI automata, then $\otimes^{\mathcal{F}} \mathcal{S} = (\prod_{i \in \mathcal{I}} Q_i, \cup_{i \in \mathcal{I}} Act_i, \delta, \prod_{i \in \mathcal{I}} I_i, (H_i)_{i \in \mathcal{I}})$ where $\delta = \{(q, x, q') \mid (q, x, q') \in \Delta_{\mathcal{S}} \wedge x \in \mathcal{F}\}$.

As the set \mathcal{F} represents component assembly of the system, we require that it contains all internal labels of former automata (that are to be composed) since the assembly binds only external services of the components. It does not concern the former internal behaviour.

Definition 4.2 We say that the automaton $\otimes^{\mathcal{F}} \{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is *defined* iff $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is a composable set of CI automata and $\mathcal{F} \supseteq \bigcup_{i \in \mathcal{I}} \mathcal{L}_{int, \mathcal{C}_i}$.

Let $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ be a composable set of CI automata, then by $\otimes \{\mathcal{C}_i\}_{i \in \mathcal{I}}$ we denote the automaton $\otimes (\bigcup_{i \in \mathcal{I}} \mathcal{L}_{\mathcal{C}_i}) \{\mathcal{C}_i\}_{i \in \mathcal{I}}$. An example of $\otimes \{\mathcal{C}_i\}_{i \in \{1,2\}}$ for automata $\mathcal{C}_1, \mathcal{C}_2$ depicted in Figure 1 is in Figure 2(a).

Example 4.3 Let \mathcal{C}_1 and \mathcal{C}_2 be the automata in Figure 1, let $\mathcal{F} = \{(2, a, 1), (1, b, 1), (1, c, 2)\}$ represent their feasible communication. As $\mathcal{F} \supseteq \mathcal{L}_{int, \mathcal{C}_1} \cup \mathcal{L}_{int, \mathcal{C}_2}$, the composite automaton $\mathcal{C}_3 = \otimes^{\mathcal{F}} \{\mathcal{C}_i\}_{i \in \{1,2\}}$ is defined. It is depicted in Figure 2(b).

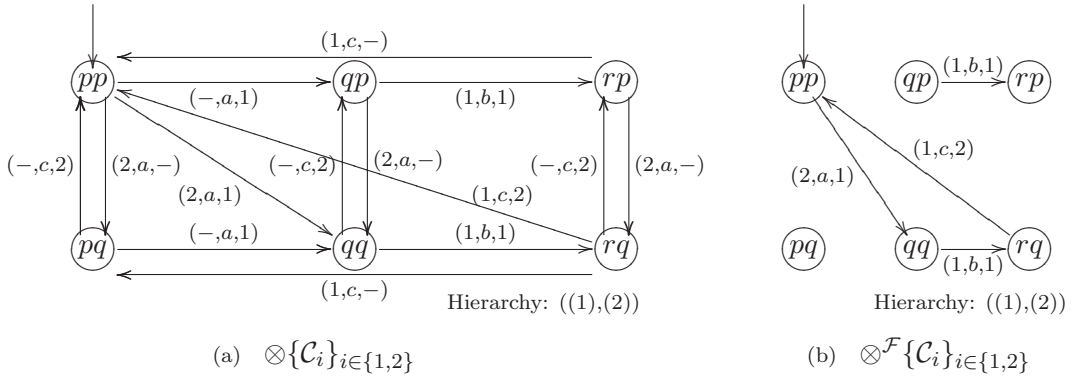


Fig. 2. Example of composite CI automata (states ij stand for (i, j))

The following lemma shows that for the operator $\otimes^{\mathcal{F}}$, the order of component automata defined by their indexes is not important (from behavioural point of view) as the composite automata resulting from different orders are equivalent.

³ By an indexed set we mean a set with an implicit linear ordering of items, given by their numerical indexes in a least to greatest manner.

⁴ It among others means that $S_{H_i}, i \in \mathcal{I}$, are pairwise disjoint and hence any component name appears in at most one automaton.

Lemma 4.4 *Let $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ be a set of CI automata, \mathcal{F} a set of labels such that the automaton $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is defined, and denote $\mathcal{L} = \bigcup_{i \in \mathcal{I}} \mathcal{L}_{\mathcal{C}_i}$. Let $f : \mathcal{I} \rightarrow \mathcal{I}$ be a bijection and for all $i \in \mathcal{I}$ denote $\mathcal{C}'_i = \mathcal{C}_{f(i)}$. Then $\otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$ is defined and*

$$\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}} \equiv_{\mathcal{L}} \otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$$

Proof. For each $i \in \mathcal{I}$, let $\mathcal{C}_i = (Q_i, Act_i, \delta_i, I_i, H_i)$ and $\mathcal{C}'_i = (Q'_i, Act'_i, \delta'_i, I'_i, H'_i)$. As the automaton $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is defined, $\mathcal{F} \supseteq \bigcup_{i \in \mathcal{I}} \mathcal{L}_{int, \mathcal{C}_i}$ and the sets S_{H_i} , $i \in \mathcal{I}$, are pairwise disjoint, $\otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$ is also defined. To prove that $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}} \equiv_{\mathcal{L}} \otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$, it suffices to show that the relation $\sim \subseteq (\prod_{i \in \mathcal{I}} Q_i) \times (\prod_{i \in \mathcal{I}} Q'_i)$ defined

$$(q_i)_{i \in \mathcal{I}} \sim (q'_i)_{i \in \mathcal{I}} \quad \text{iff} \quad \forall i \in \mathcal{I} : q_i = q'_{f(i)}$$

is an observation equivalence which satisfies $(\star\star)$ and $(\star\star\star)$ from Definition 3.1, which is straightforward. \square

Another property of $\otimes^{\mathcal{F}}$ regarding the order of composition, which is important for proofs in the next section, is the following.

Lemma 4.5 *Let $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ be a set of CI automata, \mathcal{F} a set of labels such that the automaton $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is defined, and denote $\mathcal{L} = \bigcup_{i \in \mathcal{I}} \mathcal{L}_{\mathcal{C}_i}$. Then for any $j \in \mathcal{I}$ the automaton $\otimes^{\mathcal{F}}\{\mathcal{C}_j, \mathcal{C}_{j+1}\}$ where $\mathcal{C}_{j+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I} \setminus \{j\}}$ is defined and*

$$\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}} \equiv_{\mathcal{L}} \otimes^{\mathcal{F}}\{\mathcal{C}_j, \mathcal{C}_{j+1}\}$$

Proof. For each $i \in \mathcal{I}$, let $\mathcal{C}_i = (Q_i, Act_i, \delta_i, I_i, H_i)$. Analogically to the proof of Lemma 4.4, as $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ is defined, the automata $\mathcal{C}_{j+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I} \setminus \{j\}}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_j, \mathcal{C}_{j+1}\}$ are also defined. It is again straightforward to show that the relation $\sim \subseteq (\prod_{i \in \mathcal{I}} Q_i) \times (Q_j \times \prod_{i \in \mathcal{I} \setminus \{j\}} Q_i)$ defined

$$(q_i)_{i \in \mathcal{I}} \sim (q'_j, (q'_i)_{i \in \mathcal{I} \setminus \{j\}}) \quad \text{iff} \quad \forall i \in \mathcal{I} : q_i = q'_i$$

is an observation equivalence which satisfies $(\star\star)$ and $(\star\star\star)$ from Definition 3.1. \square

It is worth saying that thanks to Lemma 3.3, Lemmas 4.4 and 4.5 hold also for any $\mathcal{L} \subseteq \bigcup_{i \in \mathcal{I}} \mathcal{L}_{\mathcal{C}_i}$. Additionally, the existence of \equiv_X between CI automata \mathcal{C} and \mathcal{C}' does not depend on labels from $X \setminus (\mathcal{L}_{\mathcal{C}} \cup \mathcal{L}_{\mathcal{C}'})$. Therefore Lemmas 4.4 and 4.5 hold also for any $\mathcal{L} \supseteq \bigcup_{i \in \mathcal{I}} \mathcal{L}_{\mathcal{C}_i}$. In general, we will write $\mathcal{C} \equiv \mathcal{C}'$ whenever $\mathcal{C} \equiv_{\mathcal{L}_{\mathcal{C}} \cup \mathcal{L}_{\mathcal{C}'}} \mathcal{C}'$, because in such cases $\mathcal{C} \equiv_X \mathcal{C}'$ for any set of labels X .

5 Substitutability of equivalent components

We would like to expect that whenever we have a system in which we replace one component with an equivalent one, the system will remain equivalent to the former one. This task is a bit more complicated, as it is parametrized by the interconnection of the component into the system (given by \mathcal{F} in $\otimes^{\mathcal{F}}$) and the accuracy of the equivalence (given by X in \equiv_X)⁵.

However, we can prove that if the two components are equivalent with respect to observable labels which encompass all their external labels (which is quite a natural condition), it is true, that if in any system with any interconnection given by

⁵ Note that X can be chosen in a way that the equivalence regards only the existing functionality of the system and allows the new component to perform additional actions that are not considered as observable (see Remark 3.2 (iii)).

some $\otimes^{\mathcal{F}}$ we replace one of the components with its equivalent, the system remains equivalent to the previous one with respect to the same set of labels X ⁶. In the following lemma we prove this for the simple case when the environment constitutes of one component only. In Theorem 5.2 we extend it to an environment created of several components, and in Corollary 5.3 we address also simultaneous replacement of several components.

Lemma 5.1 *Let $\mathcal{C}_1, \mathcal{C}_2$ and \mathcal{C}_3 be CI automata and \mathcal{F} a set of labels such that the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ are defined. Let X be a set of labels such that $X \supseteq \bigcup_{i \in \{1,2\}} \mathcal{L}_{ext, \mathcal{C}_i}$. Then*

$$\text{if } \mathcal{C}_1 \equiv_X \mathcal{C}_2 \text{ then } \otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$$

Proof. By δ_{13} we denote the transition set of automaton $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and by δ_{23} the transition set of $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$.

Because $\mathcal{C}_1 \equiv_X \mathcal{C}_2$, there is an observation equivalence of \mathcal{C}_1 and \mathcal{C}_2 with respect to X which fulfills $(\star\star)$ and $(\star\star\star)$ from Definition 3.1. Let \approx be an arbitrary, but fixed, relation satisfying these conditions.

It is necessary to show that there exists a relation, which is an observation equivalence of $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ with respect to X satisfying $(\star\star)$ and $(\star\star\star)$. We show that the relation \sim defined:

$$(p_1, p_3) \sim (q_2, q_3) \Leftrightarrow p_1 \approx q_2 \wedge p_3 = q_3 \tag{1}$$

satisfies these conditions.

a) The relation \sim fulfills $(\star\star)$ and $(\star\star\star)$.

Let $(p_1, p_3) \in I_1 \times I_3$, where I_1, I_3 are sets of initial states of $\mathcal{C}_1, \mathcal{C}_3$, then from the fact that \approx satisfies $(\star\star)$ it follows that there is a state $q_2 \in I_2$ such that $p_1 \approx q_2$ and from the definition of \sim it is seen that $(p_1, p_3) \sim (q_2, p_3)$. The relation \sim satisfies the condition $(\star\star\star)$ for similar reasons.

b) The relation \sim is an observation equivalence of $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ with respect to X .

Let $(p_1, p_3) \in Q_1 \times Q_3$, $(q_2, p_3) \in Q_2 \times Q_3$ be states such that $(p_1, p_3) \sim (q_2, p_3)$. We prove that the condition (i) from Definition 3.1 is valid. The fact, that the condition (ii) is satisfied can be proved analogically.

Suppose $((p_1, p_3), l, (p'_1, p'_3)) \in \delta_{13}$, then there are three cases to analyse:

- (1) $((p_1, p_3), l, (p'_1, p'_3))$ is caused solely by \mathcal{C}_3 , i.e. $p_1 = p'_1$ and $(p_3, l, p'_3) \in \delta_3$.
Then $(q_2, p_3), l, (q_2, p'_3) \in \text{FinPath}(\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\})$ is a path which fulfills (\star) and $(p_1, p'_3) \sim (q_2, p'_3)$.
- (2) $((p_1, p_3), l, (p'_1, p'_3))$ is caused solely by \mathcal{C}_1 , i.e. $(p_1, l, p'_1) \in \delta_1$ and $p_3 = p'_3$.
Then because $p_1 \approx q_2$ there is a path $q_2, l_1, q'_2, \dots, l_{n-1}, q'_n \in \text{FinPath}(\mathcal{C}_2)$ which fulfills (\star) and $p'_1 \approx q'_n$. Since $((p_1, p_3), l, (p'_1, p'_3)) \in \delta_{13}$ it holds that $l \in \mathcal{F}$. As $\{l_1, \dots, l_{n-1}\} \cap X \subseteq \{l\}$ and $X \supseteq \mathcal{L}_{ext, \mathcal{C}_2}$ it holds $\{l_1, \dots, l_{n-1}\} \subseteq \mathcal{L}_{int, \mathcal{C}_2} \cup \{l\}$. And since $\mathcal{L}_{int, \mathcal{C}_2} \subseteq \mathcal{F}$ and $l \in \mathcal{F}$ also $\{l_1, \dots, l_{n-1}\} \subseteq \mathcal{F}$ and thus

⁶ Note that the main strength of this proposition is that it holds for an arbitrary environment of the component and an arbitrary coordination logic given by \mathcal{F} (which may disable some behaviours of the component).

$(q_2, p_3), l_1, (q'_2, p_3), l_2, \dots, l_{n-1}, (q'_n, p_3) \in \text{FinPath}(\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\})$. From the definition it follows that the path $(q_2, p_3), l_1, (q'_2, p_3), l_2, \dots, l_{n-1}, (q'_n, p_3)$ satisfies (\star) and $(p'_1, p_3) \sim (q'_n, p_3)$.

- (3) $((p_1, p_3), l, (p'_1, p'_3))$ is caused by synchronization of \mathcal{C}_1 and \mathcal{C}_3 on $l = (n_1, a, n_2)$, i.e. $(p_1, l', p'_1) \in \delta_1$ and $(p_3, l'', p'_3) \in \delta_3$ where either $l' = (n_1, a, -)$ and $l'' = (-, a, n_2)$, or $l' = (-, a, n_2)$ and $l'' = (n_1, a, -)$.

Since $l' \in \mathcal{L}_{ext, \mathcal{C}_2}$ it is true that $l' \in X$ and because $p_1 \approx q_2$ there is a path $q_2, l_1, q'_2, \dots, l_{m-1}, q'_m, l', q'_{m+1}, \dots, l_{n-1}, q'_n \in \text{FinPath}(\mathcal{C}_2)$ such that $p'_1 \approx q'_n$ and (\star) is satisfied. Similarly to the previous case it can be shown that $\{l_1, \dots, l_{m-1}, l_{m+1}, \dots, l_n\} \subseteq \mathcal{F}$, so both $(q_1, p_3), l_1, (q'_2, p_3), \dots, l_{m-1}, (q'_m, p_3)$ and $(q'_{m+1}, p'_3), l_{m+1}, \dots, l_{n-1}, (q'_n, p'_3)$ are paths in $\text{FinPath}(\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\})$. Because $((p_1, p_3), l, (p'_1, p'_3)) \in \delta_{13}$ it holds that $l \in \mathcal{F}$. Moreover $(q'_m, l', q'_{m+1}) \in \delta_2$ and $(p_3, l'', p'_3) \in \delta_3$, consequently $((q'_m, p_3), l, (q'_{m+1}, p'_3)) \in \delta_{23}$ and thus $(q_1, p_3), l_1, (q'_2, p_3), l_2, \dots, l_{m-1}, (q'_m, p_3), l, (q'_{m+1}, p'_3), l_{m+1}, \dots, l_{n-1}, (q'_n, p'_3) \in \text{FinPath}(\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\})$.

From definition of \sim it is seen that the path satisfy (\star) and $(p'_1, p'_3) \sim (q'_n, p'_3)$. \square

Note that Lemma 5.1 does not hold for an arbitrary X as it can be seen in Figure 3 which depicts the automata $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ and the sets X, \mathcal{F} satisfying $\mathcal{C}_1 \equiv_X \mathcal{C}_2$ and not $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_2\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$.

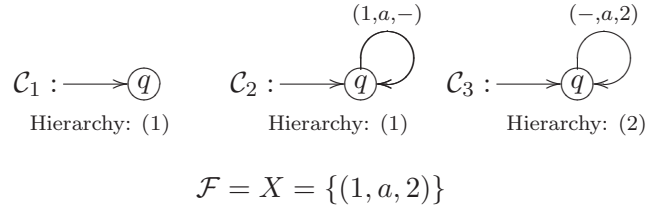


Fig. 3. Illustration that Lemma 5.1 does not hold for a general X

Lemma 5.1 can be also applied to replacement of many automata with just one automaton, one automaton with many, or a set of automata with another set. This is because a set of automata can be considered as a virtual composite automaton consisting of all (reachable) transitions of the complete transition space, as it was studied in Lemma 4.5. This idea can be applied to the following propositions too.

Now we can proceed to a more general case where the environment constitutes of several components (automata). This is the main result of this section which characterizes a precondition for safe substitutability of equivalent components in a general environment.

Theorem 5.2 *Let $\mathcal{I} \subset \mathbb{N}$, $j, k \in \mathbb{N} \setminus \mathcal{I}$, $\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{j, k\}}$ be a set of CI automata, and \mathcal{F} be a set of labels such that the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{j\}}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{k\}}$ are defined. Then for any set of labels $X \supseteq \mathcal{L}_{ext, \mathcal{C}_j} \cup \mathcal{L}_{ext, \mathcal{C}_k}$*

$$\text{if } \mathcal{C}_j \equiv_X \mathcal{C}_k \text{ then } \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{j\}} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{k\}}$$

Proof. When $\mathcal{I} = \emptyset$, the result follows directly from Lemma 5.1 and the fact that for any CI automaton \mathcal{C}'_1 and a set of labels \mathcal{F} , it holds that $\otimes^{\mathcal{F}}\{\mathcal{C}'_1\} \equiv \otimes^{\mathcal{F}}\{\mathcal{C}'_1, \mathcal{C}'_2\}$ where $\mathcal{C}'_2 = (\{q\}, \emptyset, \emptyset, \{q\}, (n))$ for a suitable $n \in \mathbb{N}$.

When $\mathcal{I} \neq \emptyset$, from Lemmas 3.3 and 4.5 it follows that

$$\begin{aligned} \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{j\}} &\equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_j, \mathcal{C}_{j+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I}}\}, \\ \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{k\}} &\equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_k, \mathcal{C}_{k+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I}}\}. \end{aligned}$$

Moreover, as $\mathcal{C}_j \equiv_X \mathcal{C}_k$, from Lemma 5.1 we get

$$\otimes^{\mathcal{F}}\{\mathcal{C}_j, \mathcal{C}_{j+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I}}\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_k, \mathcal{C}_{k+1} = \otimes\{\mathcal{C}_i\}_{i \in \mathcal{I}}\}$$

and because \equiv_X is an equivalence, $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{j\}} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I} \cup \{k\}}$. \square

The following corollary addresses the simultaneous replacement of several components at the same time.

Corollary 5.3 *Let $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ and $\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$ be sets of CI automata, and \mathcal{F} a set of labels such that the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$ are defined. Then for any set of labels $X \supseteq \bigcup_{i \in \mathcal{I}}(\mathcal{L}_{ext, \mathcal{C}_i} \cup \mathcal{L}_{ext, \mathcal{C}'_i})$*

$$\text{if } \forall i \in \mathcal{I} : \mathcal{C}_i \equiv_X \mathcal{C}'_i \text{ then } \otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$$

Proof. We can assume, without loss of generality, that $\mathcal{I} = \{1, 2, \dots, k\}$ for some $k \in \mathbb{N}$. And we use \mathcal{C}_{k+i} to denote \mathcal{C}'_i for any $i \in \mathcal{I}$. Then from Theorem 5.2 it follows that $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \dots, \mathcal{C}_k, \mathcal{C}_{k+1}\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_3, \dots, \mathcal{C}_{k+1}, \mathcal{C}_{k+2}\} \equiv_X \dots \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_{k+1}, \dots, \mathcal{C}_{k+k}\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}'_1, \mathcal{C}'_2, \dots, \mathcal{C}'_k\}$. \square

Moreover, Lemma 4.4 implies that if there exists a bijection $f : \mathcal{I} \rightarrow \mathcal{I}$ such that $\forall i \in \mathcal{I} : \mathcal{C}_i \equiv_X \mathcal{C}'_{f(i)}$ then also $\otimes^{\mathcal{F}}\{\mathcal{C}_i\}_{i \in \mathcal{I}} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$ where $\{\mathcal{C}_i\}_{i \in \mathcal{I}}$, $\{\mathcal{C}'_i\}_{i \in \mathcal{I}}$, \mathcal{F} and X follow the assumptions from Corollary 5.3.

6 Independent implementability

Until now, we have studied the problem of replacement of one component with an equivalent one. We have shown that when the components are equivalent with respect to observable actions that encompass their external communication, the resulting system remains equivalent to the former one.

In this section, we focus on another interesting issue that can be regarded using a similar concept. It is the problem of independent development of components with certainty that they can be safely reused in any system where their implementation is *compliant* to the specification stated by their environment. The solution is based on the definition of a relationship between component specification and implementation that would assure this. Such specification-implementation relation moreover allows developers to design systems as compositions of specifications and then just search for appropriate implementations. Note that the independent implementability can be regarded as the substitutability of component implementation for its specification and hence to a certain extent, we may use the results of Section 5.

However, there are several distinctions between the specification-implementation relationship and \equiv_X , which need to be taken into consideration. First, the automata representing component specification and implementation do not need to be equivalent, as the implementation may provide new services beyond those stated by the specification. Second, as the equivalence requires equality in all aspects, there is no need to discuss a potential asymmetry of required and provided services, or

of input and output actions. However we need to do this now for the specification–implementation relation. We informally consider the implementation to be compliant to the specification if it fulfills the following requirements:

Interface requirements

1. The implementation provides (resp. requires) all the services provided (resp. required) by the specification.
2. The implementation may provide (and require) services that are beyond the specification.

Behavioural requirements

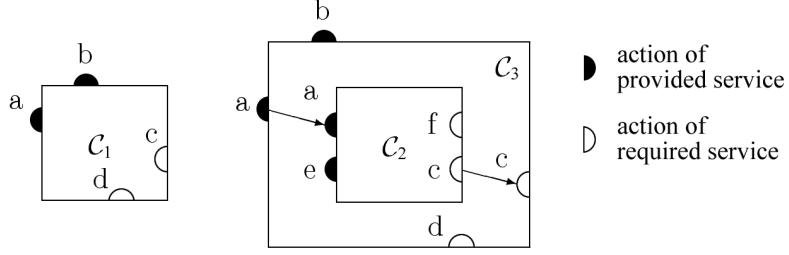
3. When serving the services provided (and required) by the specification, the implementation respects the specification in all observable steps.

Regarding (1.), the reason why the implementation must respect also required services of the specification follows from the fact that another component in the system may wait for the requisition to function correctly. The (2.) allows us to search for desired component implementation also among components that provide and require more services than the specification. The last requirement (3.) states that implementation does not require nor provide any additional observable service when serving the services provided and required by the specification.

At first glance, the symmetry between component provisions and requirements may seem surprising. However the clarification is straightforward. In fact, the asymmetry that is usually ascribed to provisions and requirements grounds deeper in the asymmetry of inputs and outputs. As provided services usually respect the pattern *input request then output response* and required services the pattern *output request then input response*, the asymmetry is forwarded to provisions and requirements too.

The systems with asymmetrical interpretation of input and output actions, as an output may be initiated anytime when an input needs to wait for a counterpart, are often called *non-blocking* systems. However, not all systems are non-blocking. *Blocking* systems, where both input and output are blocked in case a counterpart is not ready, are also of high interest. Majority of specification languages for component interactions focus on either the blocking strategy (Tracta [12], Wright [3]), or non-blocking strategy (Interface automata [9], SOFA Behavior protocols [14]). The Component-interaction automata [15] respect both as they are designed to model a variety of synchronization strategies.

We may now proceed to the definition of specification–implementation relation between two CI automata with respect to these two types of systems. In case of CI automata we may informally summarize the mentioned requirements of the relation as follows. A CI automaton \mathcal{C}_2 (implementation) is compliant to a CI automaton \mathcal{C}_1 (specification) if \mathcal{C}_2 behaves observably the same as \mathcal{C}_1 on every service provided and required by \mathcal{C}_1 . Such restriction of \mathcal{C}_2 can be formally captured using composition that encloses the component represented by \mathcal{C}_2 to a higher level component \mathcal{C}_3 as depicted in Figure 4. In the figure, the arrows symbolize delegation of services outside the component. The inner services (e, f in this case) are not accessible from outside. It means that their actions are blocked from occurring in blocking systems, but may escape (in case of outputs) in non-blocking systems.


 Fig. 4. Architectural view on restriction of \mathcal{C}_2 to given \mathcal{C}_1

Definition 6.1 Let \mathcal{C}_1 and \mathcal{C}_2 be CI automata, X be a set of labels. Then

- \mathcal{C}_2 is *compliant to \mathcal{C}_1* with respect to observable labels X in a *blocking* environment, iff $\mathcal{C}_1 \equiv_X \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ where $\mathcal{R} = \mathcal{L}_{ext,\mathcal{C}_1} \cup \mathcal{L}_{int,\mathcal{C}_2}$,
- \mathcal{C}_2 is *compliant to \mathcal{C}_1* with respect to observable labels X in a *non-blocking* environment, iff $\mathcal{C}_1 \equiv_X \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ where $\mathcal{R} = \mathcal{L}_{ext,\mathcal{C}_1} \cup \mathcal{L}_{int,\mathcal{C}_2} \cup \mathcal{L}_{out,\mathcal{C}_2}$.

We now regard the independent implementability as the substitutability of restricted component implementation $\mathcal{C}_3 = \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ for its specification \mathcal{C}_1 . As the specification–implementation relation is defined using \equiv_X , we can use the results of Section 5. In particular, if $X \supseteq (\mathcal{L}_{ext,\mathcal{C}_1} \cup \mathcal{L}_{ext,\mathcal{C}_2}) \cap \mathcal{R} = \mathcal{L}_{ext,\mathcal{C}_1} \cup \mathcal{L}_{ext,\mathcal{C}_3}$ (where \mathcal{R} is given by Definition 6.1), it follows from Lemma 5.1 that $\mathcal{C}_1 \equiv_X \mathcal{C}_3$ implies $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_4\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_3, \mathcal{C}_4\}$ for any \mathcal{C}_4 and \mathcal{F} such that the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_4\}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_3, \mathcal{C}_4\}$ are defined.

Moreover, provided that \mathcal{C}_1 is the exact specification of services used by an environment on \mathcal{C}_2 (the rest will stay unused), then it holds that, if \mathcal{C}_2 is compliant to \mathcal{C}_1 with respect to X then the system with \mathcal{C}_2 in place of \mathcal{C}_1 will be equivalent to the former one with respect to the same X . This characterization of a precondition for independent implementability is formally captured in the following lemma for the basic case when the environment constitutes of one component (automaton) only. The result can be extended for a general environment analogically to Section 5. Just before we proceed to the lemma, let us introduce an auxiliary definition needed for formalizing the concept of \mathcal{C}_1 being the exact specification of services used by an environment on \mathcal{C}_2 .

Definition 6.2 *Participate* denotes a function on sets of labels. If Y is a set of labels, then $Participate(Y)$ is a set consisting of the labels that are either from Y or represent internal communication in which the labels from Y participate. Formally, $Participate(Y) = Y \cup \{(x_1, a, x_2) \mid x_1, x_2 \in \mathbb{N} \wedge ((x_1, a, -) \in Y \vee (-, a, x_2) \in Y)\}$.

Lemma 6.3 Let $\mathcal{C}_1, \mathcal{C}_2$ and \mathcal{C}_3 be CI automata, \mathcal{R} the set given by Def. 6.1. Let \mathcal{F} be a set of labels such that $\mathcal{F} \cap Participate(\mathcal{L}_{\mathcal{C}_2} \setminus \mathcal{R}) = \emptyset$ and the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$, $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ are defined. Let X be a set of labels such that $X \supseteq (\bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,\mathcal{C}_i}) \cap \mathcal{R}$. Then

$$\text{if } \mathcal{C}_2 \text{ is compliant to } \mathcal{C}_1 \text{ w.r.t. } X^7 \text{ then } \otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$$

⁷ Generally for either blocking or non-blocking environment given by \mathcal{R} .

Proof. As $X \supseteq (\bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,C_i}) \cap \mathcal{R}$, it follows from Lemma 5.1 that $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_2\}, \mathcal{C}_3\}$. Since $\mathcal{F} \cap \text{Participate}(\mathcal{L}_{\mathcal{C}_2} \setminus \mathcal{R}) = \emptyset$, i.e. \mathcal{F} does not include labels of \mathcal{C}_2 that are not in \mathcal{R} nor their connection into new internal labels, and $\otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_2\}, \mathcal{C}_3\} \equiv \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$, it holds that $\otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_2\}, \mathcal{C}_3\} \equiv \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$. Thus $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$. \square

Let us mention one more property of the specification–implementation relation with respect to the equivalence relation defined in Section 3. For any CI automata $\mathcal{C}_1, \mathcal{C}_2$ and a set of labels $X \supseteq \bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,C_i}$ it holds that, if $\mathcal{C}_1 \equiv_X \mathcal{C}_2$ then \mathcal{C}_2 is compliant to \mathcal{C}_1 w.r.t. X in both blocking and non-blocking environment.

7 Substitutability of non-equivalent components

The results of Section 5, Lemma 5.1 and Theorem 5.2 in particular, help us to determine when we can replace one component with another one with certainty that the new system will preserve the previous behaviour. The theorem states that whenever two components are equivalent, we can replace one with the other one and the new system will be equivalent to the previous one. It can also happen that the two components are not equivalent, but the new system (with a new component in a place of the former one) is equivalent to the former system. It can be caused by the fact that the component is bound into the system in a way that some of its behaviours are disabled and that can be exactly the behaviours that distinguish the components.

The substitutability relation is defined exactly for this purpose. It extends the notion of the specification–implementation relation with the assumption that the new component \mathcal{C}_2 does not have to simulate behaviours of the former component \mathcal{C}_1 that are not used by the environment (some interfaces/services of \mathcal{C}_1 may be unused by the environment). In the following definition, the services of \mathcal{C}_1 that are really used by the environment are represented by the set E .

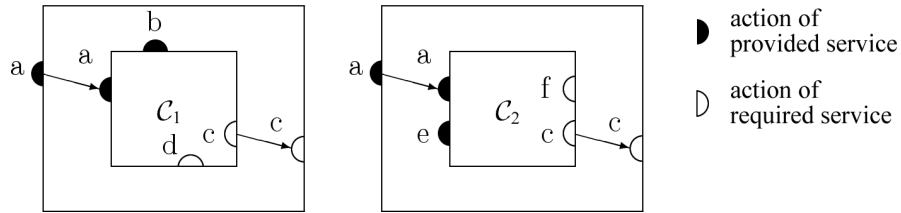


Fig. 5. Architectural view on restriction of \mathcal{C}_1 and \mathcal{C}_2 for the same environment

Definition 7.1 Let \mathcal{C}_1 and \mathcal{C}_2 be CI automata, X, E be sets of labels. Then

- \mathcal{C}_2 is *substitutable for \mathcal{C}_1* with respect to observable labels X and labels (used by the environment) E in a *blocking* environment, iff $\otimes^{\mathcal{R}}\{\mathcal{C}_1\} \equiv_X \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ where $\mathcal{R} = E \cup \mathcal{L}_{int,C_1} \cup \mathcal{L}_{int,C_2}$,
- \mathcal{C}_2 is *substitutable for \mathcal{C}_1* with respect to observable labels X and labels (used by the environment) E in a *non-blocking* environment, iff $\otimes^{\mathcal{R}}\{\mathcal{C}_1\} \equiv_X \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ where $\mathcal{R} = E \cup \mathcal{L}_{int,C_1} \cup \mathcal{L}_{int,C_2} \cup \mathcal{L}_{out,C_1} \cup \mathcal{L}_{out,C_2}$.

Similarly to the previous section, suppose that a component (a CI automaton \mathcal{C}_1) is connected with an environment (a CI automaton \mathcal{C}_3) by \mathcal{F} in such a way that the environment really uses just the services of \mathcal{C}_1 that are specified by E . Then if $X \supseteq (\mathcal{L}_{ext,\mathcal{C}_1} \cup \mathcal{L}_{ext,\mathcal{C}_2}) \cap \mathcal{R}$, and a CI automaton \mathcal{C}_2 is substitutable for \mathcal{C}_1 , i.e. $\otimes^{\mathcal{R}}\{\mathcal{C}_1\} \equiv_X \otimes^{\mathcal{R}}\{\mathcal{C}_2\}$ for \mathcal{R} given by the definition, then also the new system will be equivalent to the former one $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$.

The following lemma states this formally giving a formal characterization of safe substitutability of non-equivalent components. It is again the basic case considering one-item environment only, but extendable to the general case as in Section 5.

Lemma 7.2 *Let $\mathcal{C}_1, \mathcal{C}_2$ and \mathcal{C}_3 be CI automata, E a set of labels, and \mathcal{R} the set given by Def. 7.1. Let \mathcal{F} be a set of labels such that $\mathcal{F} \cap \text{Participate}((\mathcal{L}_{\mathcal{C}_1} \cup \mathcal{L}_{\mathcal{C}_2}) \setminus \mathcal{R}) = \emptyset$ and the automata $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and $\otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ are defined. Let X be a set of labels such that $X \supseteq (\bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,\mathcal{C}_i}) \cap \mathcal{R}$. Then*

if \mathcal{C}_2 is substitutable for \mathcal{C}_1 w.r.t. X, E ⁸ then $\otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$

Proof. As $X \supseteq (\bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,\mathcal{C}_i}) \cap \mathcal{R}$, from Lemma 5.1 we get $\otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_1\}, \mathcal{C}_3\} \equiv_X \otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_2\}, \mathcal{C}_3\}$. Hence it suffices to show that $\otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_1\}, \mathcal{C}_3\} \equiv \otimes^{\mathcal{F}}\{\mathcal{C}_1, \mathcal{C}_3\}$ and $\otimes^{\mathcal{F}}\{\otimes^{\mathcal{R}}\{\mathcal{C}_2\}, \mathcal{C}_3\} \equiv \otimes^{\mathcal{F}}\{\mathcal{C}_2, \mathcal{C}_3\}$ whenever \mathcal{F} satisfies the condition given above. This follows similarly to the proof of Lemma 6.3. \square

Note that again for any CI automata $\mathcal{C}_1, \mathcal{C}_2$, a set of labels E , and $X \supseteq \bigcup_{i \in \{1,2\}} \mathcal{L}_{ext,\mathcal{C}_i}$ it again holds that if $\mathcal{C}_1 \equiv_X \mathcal{C}_2$ then \mathcal{C}_2 is substitutable for \mathcal{C}_1 w.r.t. X, E in both blocking and non-blocking environment. It is a direct application of Theorem 5.2 for $\mathcal{I} = \emptyset$.

8 Related work

The issue of relations between components in component-based systems has already been addressed by several authors. The attention was focused mainly on the specification–implementation relation. The techniques for defining the relation are based either on strong/weak simulation or language inclusion. We have joint the first group as it allows finer grained comparison of two components because it can distinguish two behaviours that pass through different states even if their traces are the same. It comes in useful during formal verification which may also distinguish trace equivalent paths through different states.

One of the best known relations defined for component-based systems is the *refinement* relation introduced within *Interface automata* [9,8] by de Alfaro and Henzinger. It focuses on the relationship between component specification and implementation to facilitate the independent implementability of components in component-based systems. The relation respects the difference of input and output actions where each input is considered as a provision and output as a requirement of a component. According to this, an implementation A refines a specification B if each input transition of B can be simulated by A , and each output transition of A can be simulated by B . The precise definition must take into account the hidden transitions of A and B too.

⁸ Generally for either blocking or non-blocking environment given by \mathcal{R} .

A different approach to relationship of component specification and implementation was used in *compliance* relation for *SOFA Behavior protocols* [14,11] which was introduced by Plasil et al. It facilitates the decision whether the implementation given by a protocol A can replace the specification given by a protocol B without visible change for its environment. The relation is defined using two trace language inclusions. The first inclusion states that A has to render any sequence of provisions of B as it can be chosen by its environment. The second inclusion states that A leaded by the provisions of B may allow only the behaviour that was already possible for B . Additionally, A and B can be restricted to a partial alphabet if needed (to abstract from some of their actions). Besides the relation, the authors of *SOFA* also address the issue of partial bindings of component interfaces [2], which we considered in definition of our specification–implementation relation, and discuss the role of component environment that may coordinate an incorrect component in a way that the component does not perform any of its incorrect behaviours [1], which is a motivation of our substitutability relation.

Another approach that is worth mentioning was studied in [7,6] by Chaki et al. This work focuses on component substitutability directly from the verification point of view. The aim of the work is to provide an effective verification procedure that decides whether a component can be replaced with a different one without violating system correctness.

9 Conclusion and future work

We have provided a formal view on the issues of component-substitutability and independent implementability in component-based systems, which aims to facilitate the task of checking reconfiguration correctness. Based on the formalism of Component-interaction automata, we have introduced three relations which support different levels of this task: (1) equivalence relation, (2) specification–implementation relation, and (3) substitutability relation.

The first relation, which provides an important basis extended by the other two relation, is defined as an observation equivalence with respect to X which represents the set of observable labels. The concept of the set X allows us to choose the level of accuracy at which we compare the components. Moreover, as components are modelled as CI automata, we may also choose the level of correspondence among primitive sub-components of compared components (in case they are composite). It can be done using the equivalence up to renaming.

The other two relations extend the concept of equivalence relation in allowing one (specification–implementation) or both (substitutability) components to provide functionality that is not integrated into the system, and therefore should not be taken into account during component comparison. However, in non-blocking environment even the functionality that is not integrated could cause problems (output that was not awaited by the environment). Therefore we have also discussed the consequence of blocking and non-blocking environment. In addition, both specification–implementation and substitutability relations are again parametrized by a set X , and both can be viewed by way of relation up to renaming.

As one of the most significant contributions of the paper, we have stated and

proved several statements regarding the reconfiguration correctness for the equivalence relation, and discussed how these statements can be analogically applied to the other two relations. Roughly speaking, the main result shows that if we substitute a component with an equivalent one, replace a component specification with a compliant implementation, or change a component with a non-equivalent but substitutable one, and the particular relation is assessed with respect to X that includes external labels of the automata, the system remains equivalent to the former one with respect to the same X , no matter what the environment is like.

Nowadays, we study the relationship of the equivalence relation and temporal logics used for formal verification. In future, we aim to implement the equivalence checking into the verification tool DiVinE [4,10] which we have already used for verification of systems modelled as CI automata. We currently work on design of effective equivalence checking algorithms using various optimizations with respect to the input automata.

References

- [1] Adamek, J. and F. Plasil, *Erroneous architecture is a relative concept*, in: *Proceedings of the Software Engineering and Applications conference (SEA'04)* (2004), pp. 715–720.
- [2] Adamek, J. and F. Plasil, *Partial bindings of components - any harm?*, in: *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC'04)* (2004), pp. 632–639.
- [3] Allen, R. J. and D. Garlan, *The wright architectural specification language*, Technical Report CMU-CS-96-TBD, Carnegie Mellon University, School of Computer Science, USA (1996).
- [4] Barnat, J., L. Brim, I. Černá, P. Moravec, P. Ročkai and P. Šimeček, *Divine - a tool for distributed verification*, in: *Proceedings of the Computer Aided Verification conference (CAV'06)*, Seattle, WA, USA, 2006, to appear.
- [5] Brim, L., I. Černá, P. Vařeková and B. Zimmerova, *Component-Interaction automata as a verification-oriented component-based system specification*, in: *Proceedings of the ESEC/FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS'05)* (2005), pp. 31–38, published also in ACM SIGSOFT Software Engineering Notes, Volume 31, Issue 2 (March 2006).
- [6] Chaki, S., E. Clarke, N. Sharygina and N. Sinha, *Dynamic component substitutability analysis*, in: *Proceedings of the Formal Methods 2005 conference (FM'05)* (2005).
- [7] Chaki, S., N. Sharygina and N. Sinha, *Verification of evolving software*, in: *Proceedings of the ESEC/FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS'04)*, Newport Beach, California, USA, 2004, pp. 55–61.
- [8] de Alfaro, L. and T. A. Henzinger, *Interface automata*, in: *Proceedings of the 9th Annual Symposium on Foundations of Software Engineering (FSE'01)* (2001), pp. 109–120.
- [9] de Alfaro, L. and T. A. Henzinger, *Interface-based design*, in: *Proceedings of the 2004 Marktoberdorf Summer School* (2005).
- [10] *Divine - Distributed Verification Environment*, <http://anna.fi.muni.cz/divine>. URL <http://anna.fi.muni.cz/divine>
- [11] Mach, M., F. Plasil and J. Kofron, *Behavior protocols verification: Fighting state explosion*, *International Journal of Computer and Information Science* **6** (2005), pp. 22–30.
- [12] Magee, J., J. Kramer and D. Giannakopoulou, *Behaviour analysis of software architectures*, in: *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA'99)* (1999), pp. 35–50.
- [13] Milner, R., “Communication and Concurrency,” Prentice Hall, 1989.
- [14] Plasil, F. and S. Visnovsky, *Behavior protocols for software components*, *IEEE Transactions on Software Engineering* **28** (2002), pp. 1056–1076.
- [15] Vařeková, P. and B. Zimmerova, *Component-Interaction automata for specification and verification of component interactions*, in: *Proceedings of the IFM 2005 Doctoral Symposium on Integrated Formal Methods* (2005), pp. 71–75.