# Component-aware Input-Output Conformance

Alexander Graf-Brill, Holger Hermanns

# Component-aware Input-Output Conformance

Alexander Graf-Brill[1] and Holger Hermanns[1,2]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] Institute of Intelligent Software, Guangzhou, China

**Abstract** Black-box conformance testing based on a compositional model of the intended behaviour is a very attractive approach to validate the correctness of an implementation. In this context, input-output conformance is a scientifically well-established formalisation of the testing process. This paper discusses peculiar problems arising in situations where the implementation is a monolithic black box, for instance for reasons of intellectual property restrictions, while the specification is compositional. In essence, tests need to be enabled to observe progress in individual specification-level components. For that, we will reconsider input-output conformance so that it can faithfully deal with such situations. Refined notions of quiescence play a central role in a proper treatment of the problem. We focus on the scenario of parallel components with fully asynchronous communication covering very many notorious practical examples. We finally illustrate the practical implications of component-aware conformance testing in the context of a prominent example, namely networked embedded software.

**Keywords:** Model-based testing · Input-output conformance · Compositionality.

## 1 Introduction

*Component-based* or *modular systems* are systems which are composed of several components in order to provide a higher degree of functionality or just to offer the ensemble of features offered by its components. From an implementation point of view, component-based systems are very flexible since single components can be updated or exchanged easily, or the system can be extended by additional components, without having to touch the whole system.

When it comes to the verification of such systems, one usually tries to benefit from the compositional structure. Correctness of the components is easier to verify in isolation and under appropriate conditions, correctness of the whole system is derived from the correctness of all components. This reduces the overall verification effort. In particular, when updating a single component of the system, one only has to verify the new component without the need to verify the other components again. However, this approach is only applicable if the correctness properties are *compositional* [12,10,17,2,18,21,1].

Model-based testing is a validation technique where, based on a formal specification of a system, a suitable set of experiments (test suite) is generated in

an automated manner and executed on the implementation of that system, so as to assert some notion of conformance between the implementation and its specification. In model-based testing, compositional testing is a research area of its own. Given a specification and an implementation under test (IUT), each as combinations of several components, a compositional conformance tester checks conformance between the components of the specification and the respective IUT components, so as to conclude conformance between the combined specification and IUT. If this implication holds, there is no need for further integration testing when combining the different IUT components [2,7,11,6,3]. Otherwise this is a costly and time-consuming step since the combined specification has to be taken into account which is notorious in size relative to the sizes of the individual components.

What all these approaches have obviously in common is the assumption that the IUT is indeed a combination of several components which can be accessed individually. Interestingly little attention has been payed to the situation where only the specification is composed of clearly distinguished components, but the IUT is a *single* black box, i.e. a monolithic object, or an object where components are not accessible in isolation.

We consider this as a mismatch, since the previously described scenario is pretty much the norm for black-box systems protected by intellectual property rights. Especially if such systems need to undergo a certification according to some well-structured component-based or scenario-based standard. This is the concrete problem motivating our work. But beyond that there are several other reasons for attacking this challenge.

Since there are no dedicated theoretical approaches targeting the testing of such scenarios, the standard input-output conformance [23] is the natural base methodology. Input-output conformance (**ioco**) is based on the idea that a reasonable implementation of a formally specified system should assure that

> the IUT progresses as foreseen by the specification, and this progress is observable by the tester.

Since IUT progress corresponds to outputs of the IUT, this means that

1. any interaction sequence between tester and IUT possible according to the specification is followed only by IUT outputs foreseen according to the specification;
2. only in situations where no IUT output is foreseen, the IUT is allowed to be quiescent.

Quiescence, and especially the possibility to observe quiescence is a crucial ingredient to the theory of input-output conformance. It makes progress observable by refining classical testing equivalences and preorders [9,8] with concepts of refusal testing [22,19], thus enabling a more fine grained relation between systems based on their state-based capabilities to produce any output at all. In practice, quiescence is approximated by timeout mechanisms: If after some interaction sequence no IUT output is witnessed before the timeout, the IUT is interpreted

as now being quiescent. The concept of quiescence therefore provides an implicit mechanism to test for absence as well as presence of progress, without an explicit reference to real time.

This paper explores the simple question what needs to change in the theory of input-output conformance in order to be able to test for the progress of components of a component-based specification. So, we add the idea that a reasonable implementation of a component-based system should assure that

> the IUT progresses as foreseen by the *component-based* specification, and this progress is observable by the tester.

This then translates concretely to (the first item being unchanged),

1. any interaction sequence between tester and IUT possible according to the specification is followed only by IUT outputs foreseen according to the specification;
2. only in situations where no IUT output is foreseen *by some component*, the IUT is allowed to be quiescent *with respect to that component*.

The first requirement is indeed the standard **ioco** criterion considering *functional correctness* of an IUT. The second requirement is the core motivation for this paper. This requirement harvests the available information about the inner structure of a compositional specification. Since the specification is white-box, any observable behaviour of the system can be associated to the components possibly causing that behaviour. This provides the opportunity to deduce which components are taking part in an interaction, and effectively enables a fine-grained notion of quiescence. With this, we require an IUT to progress whenever possible not only on the system-level, but instead for each component of a composed system. Notably, we will apply this to monolithic black-box implementations, but we nevertheless ask them to respect the compositional nature of their specification. From a testing perspective this implies among others that we would reject an implementation which only exhibits the behaviour of a single component, in situations where other components can not stay quiescent i.e. they potentially can progress by observable behaviour. We make all this deducible by only looking at the compositional specification.

*Organisation of the paper.* After setting the stage, we first argue why precisely standard input-output conformance is ill-suited for the problem at hand. We then focus on specifications that are fully asynchronous, so they are merely collections of behaviour descriptions where none of the behaviour emerges through interaction across components. This is a very widespread scenario in practice. We explain the details of a natural solution which comes with adaptations to the quiescence definition. On the practical side we discuss in how far the resulting notions can indeed be tested for, which leads to a well-motivated revision of the theory. We finally illustrate the practical implications of component-aware conformance testing in the context of networked embedded software.

## 2 Preliminaries

The basis for model-based testing is a precise specification of the IUT which unambiguously describes what an implementation may do, respectively not do.

*Input-output transition systems.* A common semantic model to describe the behaviour of a system are labeled transition systems (LTS). In the presence of inputs and outputs, a suitable variation is provided by *Input-Output Transition Systems* (IOTS).

**Definition 1.** *An* input-output transition system *is a 5-tuple* $\langle Q, L_?, L_!, T, q_0\rangle$ *where*

- $Q$ *is a finite, non-empty set of* states*;*
- $L_?$ *and* $L_!$ *are disjoint countable sets* ($L_? \cap L_! = \emptyset$) *of* input labels *and* output labels*, respectively;*
- $T \subseteq Q \times (L \cup \{\tau\}) \times Q$*, with* $\tau \notin L$*, is the* transition relation*, where* $L = L_? \cup L_!$*;*
- $q_0$ *is the* initial state*.*

*The class of input-output transition systems with inputs in* $L_?$ *and outputs in* $L_!$ *is denoted by* $\mathcal{IOTS}(L_?, L_!)$*.*

As usual, $\tau$ represents an unobservable internal action of the system. We write $q \xrightarrow{\mu} q'$ if there is a transition labelled $\mu$ from state $q$ to state $q'$, i.e., $(q, \mu, q') \in T$. The composition of transitions $q_1 \xrightarrow{\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_{n-1}} q_n$ expresses that the system, when in state $q_1$, may end in state $q_n$, after performing the sequence of actions $\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_{n-1}$, i.e. $\exists (q_i, \mu_i, q_{i+1}) \in T, i \leq n-1$. Due to non-determinism, it may be the case, that after performing the same sequence, the system may end in another state (or multiple such states): $q_1 \xrightarrow{\mu_1 \cdot \mu_2 \cdot \ldots \cdot \mu_{n-1}} q'_n$ with $q_n \neq q'_n$.

*Traces and derived notions.* Usually an IOTS can represent the entire behaviour of a system, including concrete interactions between system and environment. One such behaviour is represented by a so-called *trace*, of which we are only interested in its observable part, obtained by abstracting from internal actions of the system. Let $p = \langle Q, L_?, L_!, T, q_0\rangle$ be an IOTS with $q, q' \in Q, L = L_? \cup L_!, a, a_i \in L$, and $\sigma \in L^*$. We write $q \overset{\epsilon}{\Longrightarrow} q'$ to express that $q = q'$ or $q \xrightarrow{\tau \cdots \tau} q'$. $q \overset{a}{\Longrightarrow} q'$ denotes the fact that $\exists q_1, q_2 \in Q : q \overset{\epsilon}{\Longrightarrow} q_1 \xrightarrow{a} q_2 \overset{\epsilon}{\Longrightarrow} q'$. This can be extended for a sequence of actions $q \overset{a_1 \cdot \ldots \cdot a_n}{\Longrightarrow} q'$ s.t. $\exists q_0, ..., q_n \in Q : q = q_0 \overset{a_1}{\Longrightarrow} q_1 \overset{a_2}{\Longrightarrow} \ldots \overset{a_n}{\Longrightarrow} q_n = q'$. $q \overset{\sigma}{\Longrightarrow}$ and $q \overset{\sigma}{\not\Longrightarrow}$ are then defined as $\exists q' : q \overset{\sigma}{\Longrightarrow} q'$ and $\nexists q' : q \overset{\sigma}{\Longrightarrow} q'$, respectively.

Furthermore, $init(q)$ denotes the set of available transitions in a state $q$, i.e., $\{\mu \in L \cup \{\tau\} \mid q \xrightarrow{\mu}\}$. The set of traces starting in state $q$ is defined as $traces(q) =_{\text{def}} \{\sigma \in L^* \mid q \overset{\sigma}{\Longrightarrow}\}$. For a given trace $\sigma$, the set of reachable states is given by the definition $q \,\mathbf{after}\, \sigma =_{\text{def}} \{q' \mid q \overset{\sigma}{\Longrightarrow} q'\}$. The extension for starting in

a set of states $Q'$ is $Q'$ **after** $\sigma =_{\text{def}} \bigcup \{q \,\mathbf{after}\, \sigma \mid q \in Q'\}$. With $der(q)$ we denote the set of all reachable states from $q$, i.e., $\{q' \mid \exists \sigma \in L^* : q \stackrel{\sigma}{\Longrightarrow} q'\}$. Following the standard literature, we restrict ourselves to *strongly convergent* IOTS i.e. there is no state that can perform an infinite sequence of internal transitions. An IOTS $p$ is called *input-enabled*, if and only if all its reachable states $q$ are input-enabled i.e. $\forall q \in der(p), \forall a \in L_?.q \stackrel{a}{\Longrightarrow}$. So, inputs can never be blocked (or come as surprises). It is common practice to work with specifications modelled as IOTS without requiring input-enabledness while IUTs are required to be represented as input-enabled IOTS. This is what we assume here, too.

*Input-output conformance and quiescence.* A specific conformance relation, input-out conformance (**ioco**) [23] dominates theoretical as well as practical work on model-based testing. It relates implementations with specifications with respect to the possible output behaviour observed after executing traces of the specification. In **ioco**, the output behaviour includes a designated output *quiescence*, abbreviated with the special label $\delta$. *Quiescence* represents the situation when there is no output to observe at all. A state $q$ is said to be quiescent, denoted by $\delta(q)$, iff $init(q) \cap (L_! \cup \{\tau\}) = \emptyset$.

Assuming $\delta \notin (L \cup \{\tau\})$ it is technically convenient to encode quiescence wherever present into the transition structure at hand. For this a *suspension automaton* $\Delta(p)$ is constructed out of an IOTS $p$, where transitions $q \stackrel{\delta}{\longrightarrow} q$ are added to any quiescent state. The set of possible outputs of a state $q$ is then defined as $out(q) =_{\text{def}} \{a \in L_! \mid q \stackrel{a}{\longrightarrow}\} \cup \{\delta \mid \delta(q)\}$, and this is lifted to sets of states $P$ by $out(P) =_{\text{def}} \bigcup\{out(q) \mid q \in P\}$. Since quiescence is now interpreted as an additional observable output, we extend the definition for traces to *suspension traces*.

**Definition 2.** *Let* $p = \langle Q, L_?, L_!, T, q_0\rangle \in \mathcal{IOTS}(L_?, L_!)$. *The* suspension traces *of* $p$ *are given by* $Straces(p) =_{\text{def}} \{\sigma \in (L \cup \{\delta\})^* \mid q_0 \stackrel{\sigma}{\Longrightarrow}\}$.

The definition of **ioco** then looks as follows:

**Definition 3.** *Given a set of input labels* $L_?$ *and a set of output labels* $L_!$*, the relation* $\mathbf{ioco} \subseteq \mathcal{IOTS}(L_?, L_!) \times \mathcal{IOTS}(L_?, L_!)$ *is defined for a specification* $s$ *and an input-enabled implementation* $i$ *as*

$$i \,\mathbf{ioco}\, s \Leftrightarrow_{\text{def}} \forall \sigma \in Straces(s) : out(i \,\mathbf{after}\, \sigma) \subseteq out(s \,\mathbf{after}\, \sigma)$$

*Partial specifications.* Since **ioco** is defined based on the suspension traces of the specification on the one hand, and only requires inclusion of the output behaviour of the IUT w.r.t. the specified outputs on the other hand, it is possible to have partial specifications. This means that an IUT does not have to implement all specified output transitions of a certain state, but these can be seen as output alternatives. Furthermore, there are no restrictions on the behaviour of an implementation once its execution left the suspension traces of the specifications i.e. it performs an *underspecified trace*. Since an underspecified trace always starts with an unspecified input action for a state input-enabled specifications do not have any underspecified trace.

*Test generation and execution.* Theoretically, a test case is a variant of an IOTS with two special trap states labeled **pass** and **fail**, whereby each other state represents all states of the specification which are reachable by the suspension trace corresponding to the trace that leads to this particular state of the test case. In order to detect quiescence, the special transition label $\theta$ is used in order to synchronise with $\delta$. A test case is then generated based on the definition of **ioco** in an iterative manner. In each iteration step there are three options. 1) For all outputs in $L_!$ and quiescence a correspondingly labelled transition is added. If the output is not foreseen by the specification, the successor state is the special state **fail** and for all valid successor states the test case generation algorithm continues in the next iteration step. 2) An input action which is enabled in one of the encoded states of the specification is chosen and a correspondingly labeled transition is added to the test case. In order to handle interrupting outputs of the IUT, corresponding transitions are added as described in 1). 3) At any iteration step, the algorithm can be stopped by placing the special state **pass**.

An execution of a test case is then the parallel composition of the test case and the IUT. A *test run* is any trace of the parallel composition which ends in a state which is labeled with **pass** or **fail**. An IUT passes a test case if and only if all possible test runs lead to states labeled with **pass**. It fails the test case otherwise. By assuming some kind of fairness, an IUT will reveal sooner or later all its nondeterministic behaviour when repeatedly executed with a test case.

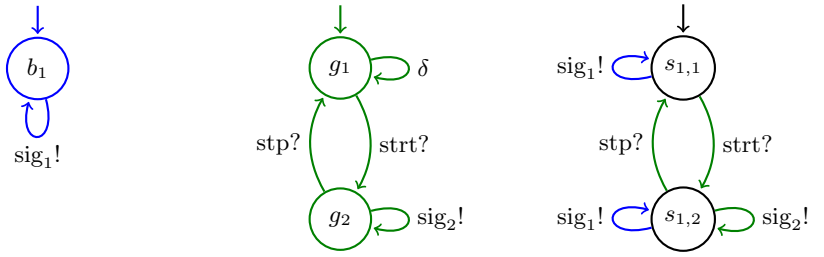
From a practical point of view, quiescence detection is realised by introducing a timer. This timer is restarted after every interaction with the IUT and upon its expiration, quiescence of the implementation is assumed and accordingly processed.

Before we delve deeper into conformance testing of component-based systems, we first give a formal definition of what we actually understand of a component-based system.

**Definition 4.** *A* component-based input-output transition system (CIOTS) *is a 6-tuple* $\langle Q, L_?, L_!, T, q_0, \boldsymbol{C}\rangle$ *where*

- *the system is the composition of* components *in the non-empty vector* $\boldsymbol{C} = \langle s_0, \ldots, s_n\rangle$ *with* $n \in \mathbb{N}_0$
- *each* $s_k \in \boldsymbol{C}$ *is a finite input-output transition system* $\langle Q_k, L_{?k}, L_{!k}, T_k, q_{0,k}\rangle \in \mathcal{IOTS}(L_{?k}, L_{!k})$
- *all components are pairwise* action-disjoint *i.e.* $\forall s_k . L_{?k} \cap \bigcup_{s_m \in \boldsymbol{C}} L_{!m} = \emptyset \wedge L_{!k} \cap \bigcup_{s_m \in \boldsymbol{C}} L_{?m} = \emptyset$
- *the sets of input labels and output labels are* $L_? = \bigcup_{s_k \in \boldsymbol{C}} L_{?k}$ *and* $L_! = \bigcup_{s_k \in \boldsymbol{C}} L_{!k}$
- *the set of states* $Q$ *is the cross product of the set of states of the components in* $\boldsymbol{C}$*, i.e.* $Q = \bigotimes_{s_k \in \boldsymbol{C}} Q_k$
- *the initial state* $q_0$ *is the cross product of the initial states of the components in* $\boldsymbol{C}$*, i.e.* $q_0 = \bigotimes_{s_k \in \boldsymbol{C}} q_{k,0}$
- *the transition relation* $T$ *is the combination of the transition relations of the components s.t.*

$$T = \{(\hat{q}_0, \ldots, \hat{q}_k, \ldots, \hat{q}_n) \xrightarrow{\mu} (\hat{q}_0, \ldots, \hat{q}'_k, \ldots, \hat{q}_n) \mid \hat{q}_k \xrightarrow{\mu} \hat{q}'_k \in T_k\}$$




(a) Component $\Delta(b)$ (b) Component $\Delta(g)$ (c) Composition $\Delta(\langle b, g\rangle)$

Figure 1: A simple component-based sensor node example.

*The class of component-based input-output transition systems with inputs in $L_?$ and outputs in $L_!$ is denoted by $\mathcal{CIOTS}(L_?, L_!)$. We say that a system $s$ is* component-based*, if and only if, $s \in \mathcal{CIOTS}(L_?, L_!)$ for some $L_?$ and $L_!$.*

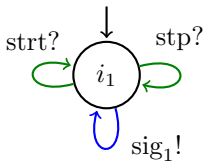
Notably, $\langle Q, L_?, L_!, T, q_0\rangle$ is itself a finite input-output transition system in $\mathcal{IOTS}(L_?, L_!)$. Since a CIOTS is already completely defined by its components, we may use the abbreviation $\langle s_0, \ldots, s_n\rangle$ in order to refer to $s = \langle Q, L_?, L_!, T, q_0, \langle s_0, \ldots, s_n\rangle\rangle$.

*Example 1.* In Figure 1 a very simple component-based specification is displayed. Figure 1a and Figure 1b show the suspension automata of the IOTSs of components $b$ and $g$, each specifying some kind of sensor. Sensor $b$ can be thought of as continuously gathering some measurement data (not modelled) which it passes on to the environment via the action $sig_1!$. Sensor $g$ works similarly using action $sig_2!$, but can be started ($strt?$) and stopped ($stp?$) from remote. Initially it is stopped – and hence quiescent.

Each IOTS $s = \langle Q, L_?, L_!, T, q_0\rangle$ can be represented as a single-component CIOTS $\hat{s} = \langle Q, L_?, L_!, T, q_0, \langle s\rangle\rangle$ and vice versa.

## 3 Conformance and component behaviour

This section provides a motivation why the well-established general **ioco** testing procedure falls short when facing a component-based specification w.r.t. the conditions we postulated in Section 1.

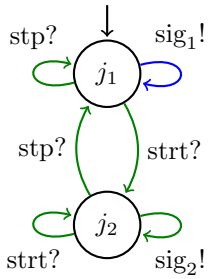



*Example 2.* A potential candidate implementation $i$ of the composed specification from Figure 1 is displayed on the right. It is input-enabled (w.r.t. the set $\{strt?, stp?\}$). Indeed, this implementation does conform to the specification, i.e. $i\,\mathbf{ioco}\langle b, g\rangle$, since $\forall\sigma \in Straces(i) : out(i_1\,\mathbf{after}\,\sigma) = \{sig_1!\}$ and $\forall\sigma' \in Straces(\langle b, g\rangle) : out(s_{1,1}\,\mathbf{after}\,\sigma')$ is either $\{sig_1!\}$ or $\{sig_1!, sig_2!\}$.

Notably, implementation $i$ does never offer the output action $sig_2!$, so the core behaviour of sensor $g$ is effectively not present in the implementation. While this omission of behaviour is perfectly legal for the theory of **ioco**, we feel that such an implementation does not exhibit the intended behaviour of two sensor nodes being combined as specified above. According to our initial reasoning, after providing the input *strt?* it is not foreseen by component $g$ to produce no output, thus implementation $i$ is not allowed to be quiescent w.r.t. component $g$.

One may object that, of course, a good compositional testing procedure should preferably be based on tests of individual components in case they do not interact with each other. Therefore one would *a priori* require that $i \,\mathbf{ioco}\, b$ as well as $i \,\mathbf{ioco}\, g$. Indeed, it turns out that $i \,\mathbf{io}\not\mathbf{co}\, g$ (due to the presence of the output action $sig_1!$, which is not foreseen by $g$) while $i \,\mathbf{ioco}\, b$. So, from this perspective, the implementation $i$ is not entirely convincing as a witness for the shortcoming of the classical **ioco** theory.

*Example 3.* Another implementation candidate $j$ is displayed on the right. It correctly outputs $sig_1!$ in the initial state and starts and stops producing the output $sig_2!$ as intended by the inputs *strt?* and *stp?*. However, the output $sig_1!$ is turned off in state $j_2$ where the output $sig_2!$ is produced only. Again it holds that $j \,\mathbf{ioco} \langle b, g \rangle$.




The essence of the problem of implementation $j$ is similar to the one of $i$ appearing in Example 2. Contrary to what we assume reasonable, some valuable output behaviour of the specification of component $b$ is not implemented, but here this is in a fragment of the state space reachable by transitions belonging to specification of component $g$. In this example, $j \,\mathbf{ioco}\, b$ as well as $j \,\mathbf{ioco}\, g$ provided we assume a suitable projection mechanism to filter the observable behaviour corresponding to the component under test.

As it stands, focusing on the behaviour of a single component, is no solution in a quest for a compositional testing theory. Instead one has to foresee arbitrary input actions of other components, in order to examine the full behaviour of an IUT. Unfortunately, "the full behaviour of an IUT" might include underspecified behaviour in case of specifications that are not input-enabled (which is not uncommon). This in turn might lead to outputs interfering with our current test run, thus, rendering such a testing approach useless, again. Hence, one has to adjust the provided inputs to the behaviour of the composed specification.

For now, we can conclude that the standard **ioco** approach is not well suited for testing scenarios involving compositional specifications. The problem is twofold. On the one hand, **ioco** is not aware of the underlying topology of the specification model. On the other hand, the relation is based on excluding unspecified output behaviour, rather than enforcing a certain output (set). We shall see that the concept of quiescence is the central leverage point regarding both aspects of the problem.

## 4 A component-aware theory

The lesson learned in the previous section is that the **ioco** approach fails at considering simultaneously the specific output behaviour of the individual components embedded in the overall behaviour of a composed specification. Especially the absence of any output from a particular component can not be detected. This is however the only indicator at hand whether or not a specified component does take part in the interactions of the IUT, or not. Thus, a quiescence definition based on the output capabilities of single components is needed.

The composition setting has similarities to the **m**ulti-**ioco** (**mioco**) relation as presented in [16] where communication with a system is assumed to occur on multiple distinct interaction *interfaces* and quiescence is then redefined s.t. each interface is associated with a dedicated quiescence action. No component structure is considered, implying that this approach is not applicable rightaway to the problem we consider. However, it serves as a strong inspriation for our approach, in which we will indeed customise the definition of **mioco** to our component-based setting. To get started, we assume an indexed family of quiescence labels of the form $\delta_k$ for $k \in \mathbb{N}$. These will serve as means to signify quiescence per individual component.

**Definition 5.** *Let $p = (p_0, \ldots, p_n)$ be a state and $P$ be a set of states of a* CIOTS *$s = \langle Q, L_?, L_!, T, q_0, \boldsymbol{C}\rangle \in \mathcal{CIOTS}(L_?, L_!)$, where $\boldsymbol{C} = \langle s_0, \ldots, s_n\rangle$ is the finite, non-empty set of component* IOTS *$s_k = \langle Q_k, L_{?k}, L_{!k}, T_k, q_{0,k}\rangle$. We define a vector $\hat{\boldsymbol{\delta}}$ (of dimension $n$) of quiescence labels by setting*

$$\hat{\delta}_k(p) =_{\text{def}} \delta(p_k) \text{ for } 0 \leq k \leq n \tag{1}$$

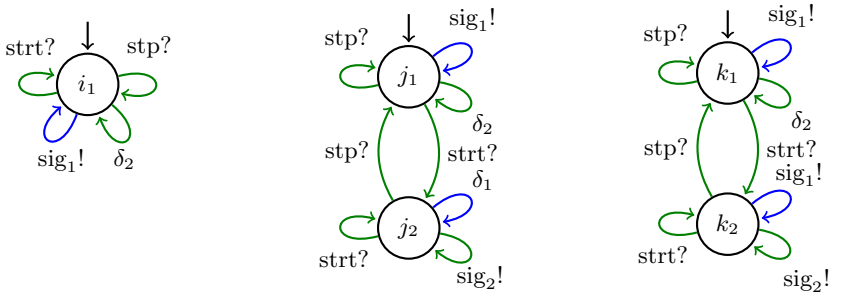
*and we propagate this into the other elements of the theory by redefining*

- $out(p) =_{\text{def}} \{x \in L_! \mid p \xrightarrow{x}\} \cup \{\delta_k \mid \hat{\delta}_k(p)\}$,
- $\Delta(s) = \langle Q, L_?, L_! \cup \{\delta_k \mid 0 \leq k \leq n\}, T \cup \{p \xrightarrow{\delta_k} p \mid p \in Q, \hat{\delta}_k(p)\}, q_0, \boldsymbol{C}\rangle$,
- $Straces(s) =_{\text{def}} \{\sigma \in (L \cup \{\delta_k \mid 0 \leq k \leq n\})^* \mid \Delta(s) \stackrel{\sigma}{\Longrightarrow}\}$.

Definition 5 provides a component-specific version of quiescence and redefines $out()$ and $Straces()$ w.r.t. this quiescence definition, where the latter one is based on the corresponding redefinition for the suspension automaton $\Delta(s)$ of a CIOTS $s$.

The behaviour of an IUT is to be interpreted relative to a given specification CIOTS, which in essence means that the output alphabets of that CIOTS induce vectors of quiescence labels for the IOTS representing the IUT, too. But since IUTs are no CIOTSs themselves, the quiescence notion $\hat{\boldsymbol{\delta}}$ from Definition 5 can not be applied directly. Thus, we need a quiescence definition and a corresponding definition of the suspension automaton for input-enabled IOTSs, given the output alphabets of interest.

**Definition 6.** *Let $p$ be a state and $P$ be a set of states of an* IOTS *$s = \langle Q, L_?, L_!, T, q_0\rangle \in \mathcal{IOTS}(L_?, L_!)$, and $\boldsymbol{L} = (L_{!0}, \ldots, L_{!n})$ be a finite vector of*




(a) implementation $i$ (b) implementation $j$ (c) implementation $k$

Figure 2: Component-based testing with **mioco**

*sets of output labels. We define a vector* $\hat{\boldsymbol{\delta}}^{\boldsymbol{L}}$ *(of dimension* $n$*) of quiescence labels by setting, for* $0 \leq k \leq n$*,*

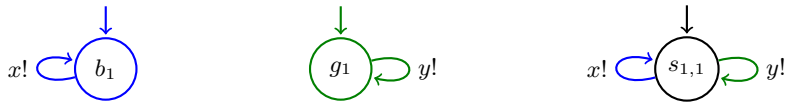
$$\delta_k^{\boldsymbol{L}}(p) =_{\text{def}} \mathit{init}(p) \cap (L_{!k} \cup \{\tau\}) = \emptyset \tag{2}$$

*and we propagate this into the definition of the suspension automaton by setting* $\Delta^{\boldsymbol{L}}(s) = \langle Q, L_?, L_! \cup \{\delta_k \mid 0 \leq k \leq n\}, T \cup \{p \xrightarrow{\delta_k} p \mid p \in Q, \delta_k^{\boldsymbol{L}}(p)\}, q_0 \rangle$*.*

*Example 4.* We revisit Figure 1 from Example 1. In the presence of Definition 5, the loop $g_1 \xrightarrow{\delta} g_1$ from component $g$ translates to a loop in $s_{1,1}$ labeled $\delta_2$ in Figure 1c. This is the only difference. Corresponding to Definition 6, implementation $i$ from Example 2 is quiescent for component $g$ in its initial state and so is implementation $j$ appearing in Example 3. In addition, the latter is quiescent for component $b$ in state $j_2$. Thus, their corresponding suspension automata for the vector of output label sets $\boldsymbol{L} = (\{\text{sig}_1!\}, \{\text{sig}_2!\})$ have loops labeled $\delta_2$ at the initial states and $\delta_1$ at $j_2$ of $\Delta^{\boldsymbol{L}}(j)$. These suspension automata are depicted in Figure 2a and Figure 2b.

For a pair (CIOTS,IOTS) of specification and implementation, Definition 6 will be used on the implementation side and Definition 5 will be used on the specification side. Both are, of course, linked by the vector of output sets $\boldsymbol{L}$. We assume the component quiescence labels to be uniquely identifiable and consistently chosen throughout the definitions, s.t. indeed the label $\delta_k$ for a particular output set $L_{!k}$ is identical in each case and matches the quiescence label for the corresponding component $s_k$ of the considered CIOTS. This enables us to drop the superscript $^{\boldsymbol{L}}$. For the remainder of this paper, we are (unless otherwise stated) working with the suspension automata without explicit reference, i.e. we use $s$ and $i$ instead of $\Delta(s)$ and $\Delta(i)$.

Based on the presented definitions and conventions, the definition for **mioco** in the context of component-based systems is as follows.




(a) component $b$ (b) component $g$ (c) composition $\langle b, g\rangle$

Figure 3: Non-deterministic encoding of parallel component behaviour.

**Definition 7.** *Given a set of input labels $L_?$ and a set of output labels $L_!$, the relation* **mioco** $\subseteq \mathcal{IOTS}(L_?, L_!) \times \mathcal{CIOTS}(L_?, L_!)$ *for input-enabled implementation $i$ and specification $s = \langle Q, L_?, L_!, T, q_0, \boldsymbol{C}\rangle$, is defined as follows:*

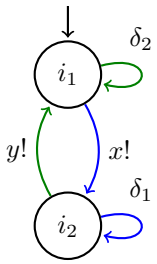
$$i \,\mathbf{mioco}\, s \Leftrightarrow_{\text{def}} \forall \sigma \in \mathit{Straces}(s) : \mathit{out}(i \,\mathbf{after}\, \sigma) \subseteq \mathit{out}(s \,\mathbf{after}\, \sigma)$$

*Example 5.* We check **mioco**-conformance of the implementation candidates depicted in Figure 2, with respect to the specification displayed in Figure 1. Implementation $i$ does not conform to $\langle b, g\rangle$ since $\mathit{out}(i \,\mathbf{after}\, \mathrm{strt?}) = \{\mathrm{sig}_1!, \delta_2\}$ which is not contained in $\mathit{out}(\langle b, g\rangle \,\mathbf{after}\, \mathrm{strt?}) = \{\mathrm{sig}_1!, \mathrm{sig}_2!\}$. Likewise, implementation $j$ is rejected again, since state $j_2$ is quiescent for component $b$ i.e. $\mathit{out}(j \,\mathbf{after}\, \mathrm{strt?}) = \{\mathrm{sig}_2!, \delta_1\}$. Contrarily, implementation $k$ **mioco**$\langle b, g\rangle$, since it implements the missing $sig_2!$ transition of implementation $j$.

## 5 A practical theory

When it comes to the practical application of **ioco**-based testing approaches, the concept of quiescence is implemented by clock timeouts and resets. In the standard **ioco** setting, a single timer for the system is sufficient. Since **mioco** is a refinement of **ioco** one can basically use the standard test generation and execution algorithms [23], but with additional quiescence timers i.e. one timer per component. The timer for a specific component is then started or reset after each observable interaction with the IUT that belongs to this component, exactly as before when considering a single global component in **ioco**. This renders **ioco** testing as a special case of **mioco** testing. There are however some subtleties in the definition of **mioco** rooted in the fact that the quiescence definition used for the IUT is state-based.

*Example 6.* In Figure 3, we have the specifications of two simple components $b$ and $g$, always producing the output $x!$ and $y!$, respectively. Their composition (shown in Figure 3c) is not quiescent for neither of the components. An implementation $i$ which simply alternates between output $x!$ and $y!$ (shown on the right) is quiescent for both components at opposite states, by definition. As a result, $i$ is not **mioco**-conformal to $\langle b, g\rangle$.




However, according to our initial motivation, there is no strong reason why such a system should be rejected. After all it does not produce any unforeseen

output (w.r.t. $L_!$) and it implements the specified behaviour for both components, by producing $x!$ and $y!$ indefinitely. And in addition, a real physical system would pass any test case that can be generated, provided sufficiently large quiescence timeout values. So we are facing a testing *practice that matches our intuition very well, but it does not match the theory.* This gap between theory and practice is rooted in the different definition bases for quiescence. Theoretically, conformance is determined by a state-based definition of quiescence. But practical black-box testing does not have information about the internal state of the IUT. As a result, quiescence detection needs to be based on the observed behaviour of the system, i.e. be trace-based. So, what we are after is a fix of the theory on the implementation side, albeit accepting the approximative nature of quiescence being implemented by timers. At the same time we want to maintain the property of our theory refining **ioco**.

The crux lies in a relaxation $\delta_k^{\boldsymbol{L}}(p)$ defined in (2) of Definition 6, so that an implementation that postpones outputs of component $k$ for some finite time will not be considered quiescent. Therefore, an IUT will be declared quiescent w.r.t. component $k$ whenever

- it is quiescent for *all* components, or
- it remains silent w.r.t. component $k$ (unless triggered by an input).

This intuition is echoed in the following definition.

**Definition 8.** *The relation* **cioco** *is defined precisely as the relation* **mioco** *in Definition 7, but on the basis of (2) of Definition 6 replaced by*

$$\delta_k^{\boldsymbol{L}}(p) =_{\text{def}} \delta(p) \vee ( \forall \sigma \in \mathit{traces}(p) : \sigma \in (L_! \cup \{\tau\})^* \Rightarrow \sigma \in ((L_! \setminus L_{!k}) \cup \{\tau\})^* \\ \wedge \quad \tau \notin \mathit{init}(p) ) \tag{3}$$

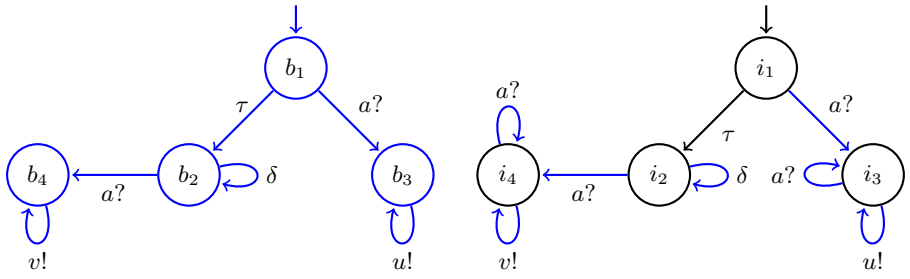
The last conjunct, enforcing non-quiescence if the implementation can step internally is needed in order to ensure that **cioco** is a conservative extension of **ioco**, as we will discuss rightaway. Standard input-output conformances considers states non-quiescent in the presence of outgoing internal transitions on both sides of the relation i.e. specification and implementation.

*Example 7.* Applying the quiescence notion to implementation $i$ from Example 6 results in a suspension automaton without any quiescent transitions i.e. both states are neither quiescent for component $b$ nor $g$. Thus, $i\,\mathbf{cioco}\langle b, g\rangle$ as intended.

**Theorem 1.** *For specification $s \in \mathcal{CIOTS}(L_?, L_!)$, and input-enabled implementation $i \in \mathcal{IOTS}(L_?, L_!)$, the following holds:*

$$i\,\mathbf{cioco}\,s \Rightarrow i\,\mathbf{ioco}\,s$$

A converse result can be established, too:




(a) specification $\langle b \rangle$ (b) implementation $i$

Figure 4: Single components considered with **ioco** vs. **cioco**.

**Theorem 2.** *For action-disjoint specification components $s_0, \ldots, s_n$ and input-enabled implementations $i_0, \ldots, i_n$ with $s_k, i_k \in \mathcal{IOTS}(L_{?k}, L_{!k})$ for $0 \leq k \leq n$, the following holds:*

$$\forall k.\ i_k \,\mathbf{ioco}\, s_k \Rightarrow i_0 \otimes \ldots \otimes i_n \,\mathbf{cioco} \langle s_0, \ldots, s_n \rangle$$

Theorem 1 and Theorem 2 together imply that **ioco** and **cioco** are equivalent for single-component specifications.

*Example 8.* We consider the single component specification $\langle b \rangle$ in Figure 4. According to **ioco**, state $b_2$ is quiescent while $b_1$ is not. Therefore the allowed outputs after the traces $a?$ and $\delta a?$ differ. The same difference applies to the states $i_2$ and $i_1$ of the input-enabled IOTS $i$ (because the definition of $\delta$ is the same on both sides) and as a result $i \,\mathbf{ioco}\, b$ holds. The above theorems ensure that $i \,\mathbf{cioco} \langle b \rangle$, too. But Theorem 2 were broken if the conjunct $\tau \notin \mathit{init}(p)$ were dropped from (3) in Definition 8. In the example, state $i_1$ would be considered quiescent (for component $b$), as well, making it indistinguishable from $i_2$, while the difference between states $b_1$ and $b_2$ would remain, because (1) in Definition 5 reduces to the classic **ioco** quiescence definition.

So, Theorem 2 hinges on the fact that internal steps are considered non-quiescent in **ioco**. While this is a useful choice on the specification side [23], it could be considered a minor shortcoming on the implementation side, where internal steps are simply unobservable. This is rooted in the fact that the same quiescence definition is used on both sides in the standard **ioco** theory. As far as we are aware, the present work is the first to break with this tradition. If one would do the same for standard **ioco**, i.e. making internal steps quiescent on the implementation side, the two theorems above could be resurrected on the basis of the following definition replacing (3).

$$\delta_k^{\boldsymbol{L}}(p) =_{\text{def}} \delta(p) \vee \forall \sigma \in \mathit{traces}(p) : \sigma \in (L_! \cup \{\tau\})^* \Rightarrow \sigma \in ((L_! \setminus L_{!k}) \cup \{\tau\})^* \tag{4}$$

We do not work out the details due to space constraints. Indeed our practical approach works independently of (4) and (3) since with blackbox testing in practice no IOTS is given.

*Application context EnergyBus.* The EnergyBus is a case study [5] which drives the concrete development of model-based testing theory and applications of us and our coworkers [15,13,14,20]. It aims at establishing a common basis for the interchange and interoperation of electric devices in the context of energy management systems (EMS). The central and innovative role of EnergyBus is the transmission and management of electrical power, in particular the safe access to electricity and its distribution inside an EnergyBus network. Conceptually, EnergyBus extends the CANopen architecture in terms of *CANopen application profiles* endorsed by the CiA association [4]. Among these, the "Pedelec Profile 1" (PP1) is very elaborate, targeting a predominant business context, which is also at the centre of ongoing international standardisation efforts as part of IEC/IS/TC69/JPT61851-3. Alongside with the standardisation, a centralised certification procedure is to be set up, according to some well-structured component-based standard. The specifications themselves are provided as informal combinations of text, protocol flow charts, data tables, and finite state automata (FSA). The definitions include several data structures and various services for e.g. initial configuration, data exchange, and basic communication capability control. The EnergyBus introduces the notion of *virtual devices*, which encapsulate the functionality of a specific, dedicated role in an EMS, e.g. of a battery pack, a motor, or a sensor unit. A real (physical) device can combine several, not necessarily different, virtual devices. For example, a public charger can be considered as being composed of a voltage converter, a secondary EBC, and a load monitoring unit, each appearing as one virtual device to the protocol.

The specification of an EnergyBus-compliant system resembles a hierarchical, tree-like structure as sketched in Figure 5. On top of this schematic structure, technically, a CANopen device might even consist of several EnergyBus devices. A real physical device (*field device*) might incorporate several CANopen device. The EnergyBus specification is a compositional specification by design. A generic CANopen and EnergyBus device has to deliver several services and continuously transmit contemporary runtime data. Furthermore, depending on the actually implemented virtual device(s), additional services have to be provided. All these protocols are concurrently running and are loosely coupled top-down and horizontally inside of a device. Depending on the chosen degree of abstraction, the involved components are purely asynchronous which holds especially for all device-level abstraction layers.

The theory developed in this paper improves the value of model-based testing in the EnergyBus context to an enormous extent. At the **cioco** core is a dedicated management of individual timers per specification-level component. The timer for a specific component is started or reset after each observable interaction with the IUT that belongs to this component, just as discussed for **mioco** in the beginning of Section 5, but now with the matching theoretical underpinning. The concrete difference to the standard, component-unaware testing approach

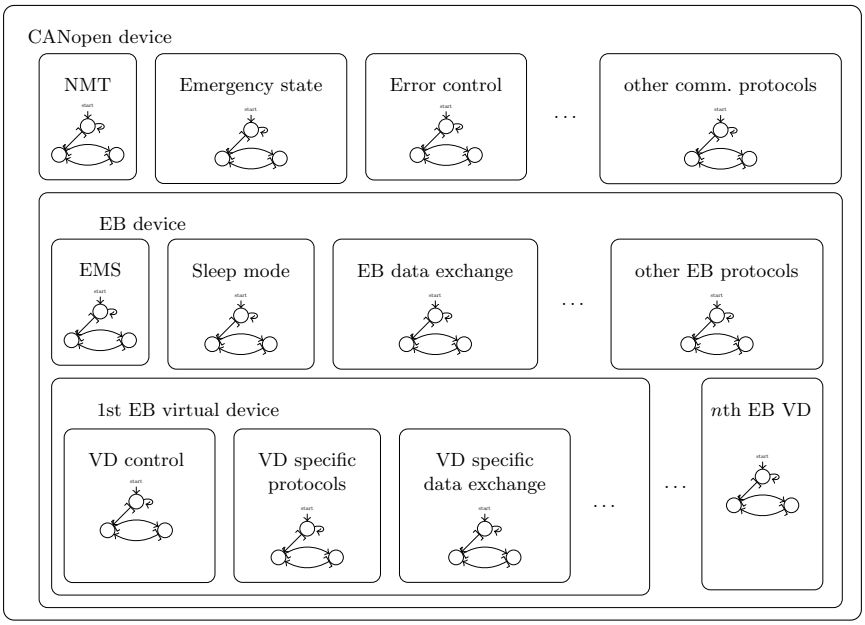



Figure 5: Schematic view on the specification of an EnergyBus (EB) device

is that the latter is unable to cope with problems as simple as some components just stopping to work. Without our theory extension, these problems induce the need for a careful manual inspection of each seemingly passing test run, in order to double check for the absence of such unwanted behaviour. By using the presented component-aware approach of this paper, all needed checks are done mechanically, on the basis of timer mechanisms that are fully automated.

As an intermediate step, one can use the theory developed to justify a sound transfer of the observation mechanism of the critical circumstances from the tester to the adapter component, sitting between the actual IUT and the tester. The adapter is then equipped with a series of observer processes which maintain dedicated timers for each specific transmission expected from the IUT. The timers are started and stopped, according to the specification knowledge transferred into the adapter component. In case a time-out occurs, the adapter sends a special *absenceOfX!* output to the tester. Observing that output makes the tester stop with a **fail**-verdict. Indeed we performed our empirical evaluation succesfully using this approach, too. It however has the drawback that the adapter component needs to be tailored to each case study variation manually. This is not needed if instead implementing the **cioco**-testing mechanisation as described.

## 6 Conclusion

This paper has developed a component-aware, yet conservative, extension of model-based input-output conformance testing. Some effort has gone into making this theory practical, which is linked to asymmetric definitions of component quiescence for specification and implementation.

Our exclusive focus has been on components that do not communicate by synchronisation. However, there are many real world examples where components of a system are meant to exchange information, which is usually modelled by synchronisation over shared or complementary actions. The theory we presented can be extended in this direction, subject to several design choices which for space constraint reasons we can only briefly touch upon here: (1) Synchronisation between components might be internally hidden and its effects may be observable in the behaviour of another component. (2) Component-local progress via synchronisation needs a synchronisation partner which may or may not be available or may preempt a possible synchronisation by alternative transitions. (3) While the standard **ioco** theory prohibits internal transition cycles, these arise rather natural when considering synchronising components. (4) From a practical point of view, this extended scenario introduces several additional challenges because a timer reset does no longer directly correlate with observable behaviour of a single component.

*Acknowledgements.* This work has received financial support by the ERC Advanced Investigators Grant 695614 (POWVER) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 389792660 as part of TRR 248, see https://perspicuous-computing.science.

## References

1. Benes, N., Daca, P., Henzinger, T.A., Kretínský, J., Nickovic, D.: Complete composition operators for ioco-testing theory. In: Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering, CBSE 2015, Montreal, QC, Canada, May 4-8, 2015. pp. 101–110 (2015). https://doi.org/10.1145/2737166.2737175
2. van der Bijl, M., Rensink, A., Tretmans, J.: Compositional testing with ioco. In: Petrenko, A., Ulrich, A. (eds.) Formal Approaches to Software Testing, Third International Workshop on Formal Approaches to Testing of Software, FATES 2003, Montreal, Quebec, Canada, October 6th, 2003. LNCS, vol. 2931, pp. 86–100. Springer (2003). https://doi.org/10.1007/978-3-540-24617-6_7
3. Braspenning, N.C.W.M., van de Mortel-Fronczak, J.M., Rooda, J.E.: A model-based integration and testing method to reduce system development effort. Electr. Notes Theor. Comput. Sci. **164**(4), 13–28 (2006). https://doi.org/10.1016/j.entcs.2006.09.003
4. CAN in Automation Int. Users and Manufacturers Group e.V.: CiA 301 CANopen Application Layer and Comm. Profile, v. 4.2.0 (February 2011)
5. CAN in Automation Int. Users and Manufacturers Group e.V., EnergyBus e.V.: CiA 454 Draft Standard Proposal Application profile for energy management systems – doc. series 1-14, v. 2.0.0 (June 2014)

6. Carver, R.H., Lei, Y.: A modular approach to model-based testing of concurrent programs. In: Lourenço, J., Farchi, E. (eds.) Multicore Software Engineering, Performance, and Tools - International Conference, MUSEPAT 2013, St. Petersburg, Russia, August 19-20, 2013. Proceedings. LNCS, vol. 8063, pp. 85–96. Springer (2013). https://doi.org/10.1007/978-3-642-39955-8_8
7. Daca, P., Henzinger, T.A., Krenn, W., Nickovic, D.: Compositional specifications for ioco testing. In: Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA. pp. 373–382. IEEE Computer Society (2014). https://doi.org/10.1109/ICST.2014.50
8. De Nicola, R.: Extensional equivalences for transition systems. Acta Inf. **24**(2), 211–237 (1987). https://doi.org/10.1007/BF00264365
9. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. **34**, 83–133 (1984). https://doi.org/10.1016/0304-3975(84)90113-0
10. Garavel, H., Lang, F., Mateescu, R.: Compositional verification of asynchronous concurrent systems using CADP. Acta Inf. **52**(4-5), 337–392 (2015). https://doi.org/10.1007/s00236-015-0226-1
11. Gotzhein, R., Khendek, F.: Compositional testing of communication systems. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) Testing of Communicating Systems, 18th IFIP TC6/WG6.1 International Conference, TestCom 2006, New York, NY, USA, May 16-18, 2006, Proceedings. LNCS, vol. 3964, pp. 227–244. Springer (2006). https://doi.org/10.1007/11754008_15
12. Graf, S., Steffen, B., Lüttgen, G.: Compositional minimisation of finite state systems using interface specifications. Formal Asp. Comput. **8**(5), 607–616 (1996). https://doi.org/10.1007/BF01211911
13. Graf-Brill, A., Hartmanns, A., Hermanns, H., Rose, S.: Modelling and certification for electric mobility. In: 15th IEEE International Conference on Industrial Informatics, INDIN 2017, Emden, Germany, July 24-26, 2017. pp. 109–114. IEEE (2017). https://doi.org/10.1109/INDIN.2017.8104755
14. Graf-Brill, A., Hermanns, H.: Model-based testing for asynchronous systems. In: Petrucci, L., Seceleanu, C., Cavalcanti, A. (eds.) Critical Systems: Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18-20, 2017, Proceedings. LNCS, vol. 10471, pp. 66–82. Springer (2017). https://doi.org/10.1007/978-3-319-67113-0_5
15. Graf-Brill, A., Hermanns, H., Garavel, H.: A model-based certification framework for the energybus standard. In: Ábrahám, E., Palamidessi, C. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings. LNCS, vol. 8461, pp. 84–99. Springer (2014). https://doi.org/10.1007/978-3-662-43613-4_6
16. Heerink, L.: Ins and Outs in Refusal Testing. Ph.D. thesis, University of Twente, Enschede, Netherlands (1998)
17. Henzinger, T.A., Qadeer, S., Rajamani, S.K.: You assume, we guarantee: Methodology and case studies. In: Hu, A.J., Vardi, M.Y. (eds.) Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings. LNCS, vol. 1427, pp. 440–451. Springer (1998). https://doi.org/10.1007/BFb0028765

18. Janssen, R., Tretmans, J.: Matching Implementations to Specifications: The Corner Cases of *ioco*. In: ACM/SIGAPP Symp. on Applied Computing – Software Verification and Testing Track. pp. 2196–2205. ACM, USA (2019), https://sumbat.cs.ru.nl/Publications
19. Langerak, R.: A testing theory for LOTOS using deadlock detection. In: Brinksma, E., Scollo, G., Vissers, C.A. (eds.) Protocol Specification, Testing and Verification IX, Proceedings of the IFIP WG6.1 Ninth International Symposium on Protocol Specification, Testing and Verification, Enschede, The Netherlands, 6-9 June, 1989. pp. 87–98. North-Holland (1989)
20. Marsso, L., Mateescu, R., Serwe, W.: TESTOR: A modular tool for on-the-fly conformance test case generation. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II. LNCS, vol. 10806, pp. 211–228. Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_13
21. Noroozi, N., Mousavi, M.R., Willemse, T.A.C.: Decomposability in input output conformance testing. In: Proceedings Eighth Workshop on Model-Based Testing, MBT 2013, Rome, Italy, 17th March 2013. pp. 51–66 (2013). https://doi.org/10.4204/EPTCS.111.5
22. Phillips, I.: Refusal testing. Theor. Comput. Sci. **50**, 241–284 (1987). https://doi.org/10.1016/0304-3975(87)90117-4
23. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers. LNCS, vol. 4949, pp. 1–38. Springer (2008). https://doi.org/10.1007/978-3-540-78917-8_1