Component Interfaces with Loosely Synchronous Communication

Raimund Kirner, Simon Maurer, Olga Tveretina
School of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
{r.kirner, s.maurer, o.tveretina}@herts.ac.uk

Abstract

Interface automata have been introduced as a way to advance from value and domain descriptions of type systems to temporal interface descriptions. The original introduction of interface automata used a notion of buffered communication with infinite buffer size. This communication model is suitable to abstract the behaviour of many computing aspects with asynchronous communication between components.

In this paper we present Loosely Synchronous Interface Automata *(LSIA) to describe interfaces of components with loosely synchronised communication. Loosely synchronised communication facilitates a blocking semantics of communication for both sender and receiver. With loosely synchronisation it is possible to describe systems where a precise order of events is necessary. For example, cyber-physical systems often include control tasks where the exact order of events is necessary for a safe operation. With LSIAs it is possible to check compatibility of interface models including safety-relevant properties like lifeness. In this paper we describe the composition of LSIAs and show examples.*

1 Introduction

Safety-critical systems require methods to assure their correct and reliable behaviour. When developing a system based on multiple components, a method is needed to ensure that these components can correctly work together. Type systems have been used with the first typed programming languages to ensure their the static compatibility of components in value and domain. Later, *interface automata* have been introduced by de Alfaro et al. as a way to reason about the compatibility of system components in the temporal domain [3]. The theory of interface automata allows to specify the interfaces multiple system components and verify their compatibility when combined. The interface automata as defined by de Alfaro et al. focuses on asynchronous communication, i.e., the sending of a message never blocks, as there is assumed an unbounded communication buffer between sender and receiver of a message. To achieve size reduction in the combination of interface automata, de Alfaro et al. also defined a special compatibility semantics, where two components are assumed to be compatible as long as there exists at least one possible environment for them to correctly operate [3].

Similar to interface automata, multisession types have been itroduced in 2008 by Yoshida et al. to formally describe the temporal interaction between multiple components [6]. Similar to the original interface automata, multisession types also focus on an asynchronous communication semantics [2].

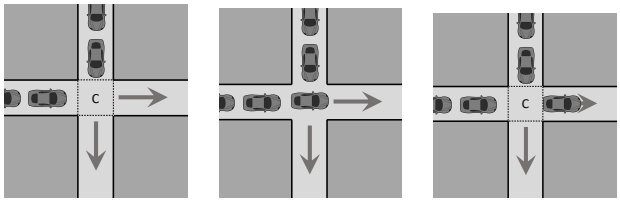
Tripakis et al. introduced interface theories for synchronous communication semantics in 2011 [8]. The basic concept of synchronous communication semantics is that the communication is aligned with some form external clock. An example of systems with synchronous communication semantics are time-triggered communication systems, which have been introduced to build ultra-dependable control systems [7]. The interface theory of synchronous communication semantics has been also applied to time-triggered systems [5].

In this paper we focus on systems with a loosely synchronous communication semantics, i.e., communication semantics where both the send and the receive of a message are blocking. With loosely synchronous communication we can have components that progress independently of each other, but get synchronised at the time of communication. This way we can specify control systems where we need a safe logic order of activities but not necessarily an alignment to an external

978-1-7281-9690-9/20/$31.00

clock base as it would be the case with synchronous communication.

In this paper we introduce the *Loosely Synchronous Interface Automata* (LSIA) to model systems with loosely synchronous communication. While it would be possible to use for our LSIA also the optimistic notion of interface compatibility as defined by de Alfaro et al., we focus in this paper on the interface combination that de Alfaro et al. call traditional pessimistic approach, i.e., where two components are assumed to be compatible only if they operate correctly regardless of the environment. To make the distinction between these two possible semantics clear, we use the term interface compatibility in the same optimistic meaning as introduced by de Alfaro et al. and define interfaces to be *harmonic* if there they operate correctly together regardless of the environment.

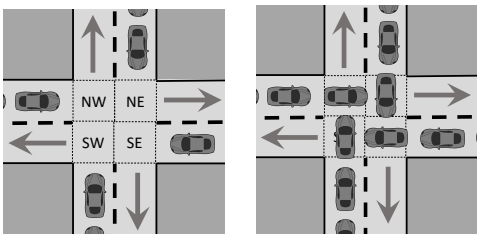


(a) (b) (c)

Figure 1: Example 1: Crossing of One-Lane Roads

To give an intuitive example of how LSIA can be used, we assume a street crossing of two one-lane roads as shown in Figure 1. To simplify things on both roads cars drive only straight, no turning at the crossing. One way to control the flow of cars in such a crossing would be to use synchronous communication, where the flow of cars is controlled by an external clock, aka traffic light. With LSIA we can model such a street crossing based on demand, aka a traffic light system that does not switch based on a fixed schedule, but rather adapt itself to the demand on each street. In case that both streets have demand, as shown in Figure 1(a), one of them would given priority. With LSIA such a choice is non-deterministic, while a real implementation of systems would have to choose some fairness policy. Figure 1(b) shows the case of a car placed at the crossroad section, blocking other cars till it leaves the crossroad as in Figure 1(c).

To show more interesting system properties, we take four instances of the example shown in Figure 1 and merge them into one crossing with two dual-line roads, as shown in Figure 2(a). In this merged example, we have now four different crossraod sections (NW, NE, SW, SE) and a car needs to allocate two of them subsequently to eventually pass the crossing. Specifying the



(a) (b)

Figure 2: Example 2: Crossing of Dual-Lane Roads

LSIA of the components one can express the semantics of the combined system. For example, as shown in Figure 2(b), an implementation of this example could lead to a deadlock [1] (a form of permanent blocking), if not designed carefully. To express possible deadlocks in the interface combination, the harmonicity of interfaces is important, rather than the compatibility of interfaces.

The remainder of this paper is structured as follows: in Section 2 we discuss process networks with loosely synchronous communication, Loosely Synchronous Interface Automaton (LSIA) is introduced in Section 3, and Section 4 concludes the paper.

2 Process Networks with Loosely Synchronous Communication

The communication model of process networks described in this paper is based on loosely synchronous communication. It referred later as *Process Network with Loosely Synchronous Communication (PNLSC).* A PNLSC Π consists of a set of processes $\mathcal{PN}$, where each process interacts with other processes via its input and output ports.

Definition 1 (Process)**.** *Formally, a process $\mathcal{N}$ is defined as a tuple*

$$\mathcal{N} = \langle \mathcal{P}^I_{\mathcal{N}}, \mathcal{P}^O_{\mathcal{N}} \rangle,$$

where

- $\mathcal{P}^I_{\mathcal{N}}$ *is a finite set of input ports of process* $\mathcal{N}$*;*
- $\mathcal{P}^O_{\mathcal{N}}$ *is a finite set of output ports of process* $\mathcal{N}$*;*
- $\mathcal{P}_{\mathcal{N}}$ *is the* signature *of* $\mathcal{N}$*, and it is defined as* $\mathcal{P}_{\mathcal{N}} = \mathcal{P}^I_{\mathcal{N}} \cap \mathcal{P}^O_{\mathcal{N}}$ *with* $\mathcal{P}^I_{\mathcal{N}} \cup \mathcal{P}^O_{\mathcal{N}} = \emptyset$*, that is the ports of these two port sets have to be mutually distinctive.*

Since each process has type Multiple-Input Multiple-Output (MIMO), it holds that $|\mathcal{P}^I_N| \geq 0$ and $|\mathcal{P}^O_N| \geq 0$. Note, that a process can have a persistent state and, thus, is not necessarily functional.

environment provides the corresponding action, while non-blocking actions can be processed immediately.

Note that LSIAs are input and output deterministic (hidden actions do not require determinism), that is, for any $a \in (\mathcal{A}^I \cup \mathcal{A}^O)$ and any $s_j, s_k \in \mathcal{S}$,

$$\langle s_i, a, s_j\rangle, \langle s_i, a, s_k\rangle \in \delta \implies s_j = s_k$$

Above condition means that from one interface state any action $a \in (\mathcal{A}^I \cup \mathcal{A}^O)$ can be part of at most one outgoing transition. An action $a \in \mathcal{A}$ is called *enabled* in a state $s \in \mathcal{S}$ if and only if the condition defined by Equation (4) is satisfied.

$$\exists s' \in \mathcal{S} : \langle s, a, s'\rangle \in \delta \tag{4}$$

It might be desirable for a protocol to reach a certain state in which it will reside indefinitely. Hence, we define a set of *end states* $\mathcal{S}^{end}$ of a Synchronous Interface Automaton (SIA) below in (5).

$$\mathcal{S}^{end} = \{s \in \mathcal{S} \mid \forall s' \in \mathcal{S}, \forall a \in \mathcal{A} : \langle s, a, s'\rangle \notin \delta\} \tag{5}$$

Informally, an end state is a state where no further transition, triggered by any action, is possible. Note that this holds for any action, i.e. no distinction is made between different action types.

As two processes can share ports, their corresponding LSIAs can share actions. In fact, for two processes $\mathcal{M}$ and $\mathcal{N}$ to interact, their corresponding LSIA $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ must have *shared* actions. Shared actions are defined in Equation (6) which is similar to the definition of shared ports provided by Equation (1).

$$shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) = (\mathcal{A}^I_{\overline{\mathcal{M}}} \cap \mathcal{A}^O_{\overline{\mathcal{N}}}) \cup (\mathcal{A}^O_{\overline{\mathcal{M}}} \cap \mathcal{A}^I_{\overline{\mathcal{N}}}) \tag{6}$$

From Equations (1), (11), and (12), it follows that

$$shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) \subseteq shared_{\mathcal{P}}(\mathcal{M}, \mathcal{N}), \tag{7}$$

which means that not all ports of the process interface have to be used by its LSIA. The set of actions that is excluded from the set of shared actions is called *ignored* actions and is defined by Equation (8).

$$ignored_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) = shared_{\mathcal{P}}(\mathcal{M}, \mathcal{N}) \setminus shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) \tag{8}$$

All input and output actions of the LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ that do not correspond to the set of shared ports of the processes $\mathcal{M}$ and $\mathcal{N}$ are called *open* actions. Open actions are defined below by Equation (9).

$$open_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) = (\mathcal{A}^I_{\overline{\mathcal{M}}} \cup \mathcal{A}^I_{\overline{\mathcal{N}}} \cup \mathcal{A}^O_{\overline{\mathcal{M}}} \cup \mathcal{A}^O_{\overline{\mathcal{N}}}) \setminus shared_{\mathcal{P}}(\mathcal{M}, \mathcal{N}) \tag{9}$$

3.1 Composition of LSIAs

In this section we formally define the composition of two LSIAs. As described in Section 1, there are different ways to specify whether interfaces work together correctly. As introduced by de Alfaro et al. in [4], compatibility of interfaces requires that there exists at least one environment where these interfaces work correctly. We call interfaces to be *harmonic* if these interfaces work together correctly in all possible environments. This harmonicity of interfaces is needed when one wants to exclude behaviour like permanent blocking as shown in Figure 2(b).

To test for harmonicity of interfaces, we can use their composition that preserves the full behaviour of each interface. Two LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ can be composed into a combined LSIA $\overline{\mathcal{M}\mathcal{N}}$ if their actions are non-conflicting as it is defined below:

$$\begin{aligned}
\mathcal{A}^I_{\overline{\mathcal{M}}} \cap \mathcal{A}^I_{\overline{\mathcal{N}}} &= \emptyset \\
\mathcal{A}^O_{\overline{\mathcal{M}}} \cap \mathcal{A}^O_{\overline{\mathcal{N}}} &= \emptyset \\
\mathcal{A}_{\overline{\mathcal{M}}} \cap \mathcal{A}^H_{\overline{\mathcal{N}}} &= \emptyset \\
\mathcal{A}^H_{\overline{\mathcal{M}}} \cap \mathcal{A}_{\overline{\mathcal{N}}} &= \emptyset
\end{aligned}$$

Note that actions can be renamed to solve such conflicts.

Definition 4 (LSIA composition operator $\otimes$)**.** *The composition of two LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ into a LSIA $\overline{\mathcal{M}\mathcal{N}} = \overline{\mathcal{M}} \otimes \overline{\mathcal{N}}$ is defined as a tuple*

$$\overline{\mathcal{M}\mathcal{N}} = \overline{\mathcal{M}} \otimes \overline{\mathcal{N}} = \langle \mathcal{S}, s, \mathcal{A}^I, \mathcal{A}^O, \mathcal{A}^H, \delta\rangle$$

where

$$\begin{aligned}
\mathcal{S} &= \mathcal{S}_{\overline{\mathcal{M}}} \times \mathcal{S}_{\overline{\mathcal{N}}} \\
s &= \langle s^0_{\overline{\mathcal{M}}}, s^0_{\overline{\mathcal{N}}}\rangle \\
\mathcal{A}^I &= (\mathcal{A}^I_{\overline{\mathcal{M}}} \cup \mathcal{A}^I_{\overline{\mathcal{N}}}) \setminus \mathcal{S}\mathcal{I} \\
\mathcal{A}^O &= (\mathcal{A}^O_{\overline{\mathcal{M}}} \cup \mathcal{A}^O_{\overline{\mathcal{N}}}) \setminus \mathcal{S}\mathcal{I} \\
\mathcal{A}^H &= \mathcal{A}^H_{\overline{\mathcal{M}}} \cup \mathcal{A}^H_{\overline{\mathcal{N}}} \cup shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})
\end{aligned}$$

with $\mathcal{S}\mathcal{I} = shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) \cup ignored_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})$, and $shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})$ and $ignored_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})$ defined by Equation (6) and Equation (8) respectively.

A transition $\big\langle\langle s_m, s_n\rangle, a, \langle s'_m, s'_n\rangle\big\rangle \in \delta$ if for a transition $\langle s_m, a, s'_m\rangle \in \delta_{\overline{\mathcal{M}}}$ and a transition $\langle s_n, a, s'_n\rangle \in \delta_{\overline{\mathcal{N}}}$, the following holds:

$$\begin{aligned}
&(a \in shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})) \\
&\quad \vee (a \notin shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) \wedge s_n = s'_n) \\
&\quad \vee (a \notin shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}}) \wedge s_m = s'_m)
\end{aligned} \tag{10}$$

By composing two LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ into the resulting LSIA $\overline{\mathcal{MN}}$, shared actions of $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ become internal actions of $\overline{\mathcal{MN}}$. Together with the internal actions of each LSIA they form the set of internal actions of $\overline{\mathcal{MN}}$:

$$\mathcal{A}^H_{\overline{\mathcal{MN}}} = \mathcal{A}^H_{\overline{\mathcal{M}}} \cup \mathcal{A}^H_{\overline{\mathcal{N}}} \cup shared_{\mathcal{A}}(\overline{\mathcal{M}}, \overline{\mathcal{N}})$$

Note that ignored actions of the LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ are not propagated to the resulting LSIA $\overline{\mathcal{MN}}$.

The composition of two LSIAs as it is defined in Definition 4 creates a composed process from the corresponding processes as it is defined in Definition 2.

3.2 Relation of a LSIA to its Process

A LSIA describes the protocol behaviour of a process, hence there is a direct relation between the set of ports $\mathcal{P}_{\mathcal{N}}$ of a process $\mathcal{N}$ and the set of actions $\mathcal{A}_{\overline{\mathcal{N}}}$ of its LSIA $\overline{\mathcal{N}}$.

An output action $a_i \in \mathcal{A}^O_{\overline{\mathcal{N}}}$ and an input action $a_j \in \mathcal{A}^I_{\overline{\mathcal{N}}}$, each labelling a transition of LSIA $\overline{\mathcal{N}}$, represent the ability of the protocol of a process $\mathcal{N}$ to write a message to the environment via an output port $a_i \in \mathcal{P}^O_{\mathcal{N}}$ or read a message from the environment via an input port $a_j \in \mathcal{P}^I_{\mathcal{N}}$, respectively. Note that input and output actions only represent the ability of the protocol to perform the action, whether it is possible to do so depends on the environment. This is due to the semantics of synchronous communication where both communication partners need to be ready for a transmission in order to transmit a message. Every input action $a_i \in \mathcal{A}^I_{\overline{\mathcal{N}}}$ (output action $a_j \in \mathcal{A}^O_{\overline{\mathcal{N}}}$) of the input alphabet $\mathcal{A}_{\overline{\mathcal{N}}}$ of LSIA $\overline{\mathcal{N}}$ has a corresponding input port $a_i \in \mathcal{P}^I_{\mathcal{N}}$ (output port $a_j \in \mathcal{P}^O_{\mathcal{N}}$) in the signature $\mathcal{P}_{\mathcal{N}}$ of process $\mathcal{N}$. However, not every port is necessarily used by a specific protocol description. Hence,

$$\mathcal{A}^I_{\overline{\mathcal{N}}} \subseteq \mathcal{P}^I_{\mathcal{N}} \quad (11)$$

$$\mathcal{A}^O_{\overline{\mathcal{N}}} \subseteq \mathcal{P}^O_{\mathcal{N}} \quad (12)$$

Internal actions are not related to the ports of a process as they happen independently of the environment the process is placed in.

It follows from the relation given by (7), that even though two processes $\mathcal{M}$ and $\mathcal{N}$ might share ports, their corresponding LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ need not necessarily describe this interaction, i.e. the process implementation does not rely on ignored ports.

Figure 3 depicts a process $\mathcal{N}_1$ where the interaction protocol is described by a LSIA $\overline{\mathcal{N}}_1$. States are represented as nodes, transitions as directed edges where the label of an edge is the name of an action. Input actions are marked with a question mark '?', output actions are marked with an exclamation mark '!', and internal actions are marked with a semicolon ';'. Note that in the example of Figure 3 port $a2 \in \mathcal{P}_{\mathcal{N}_1}$ has no corresponding action in LSIA $\overline{\mathcal{N}}_1$.

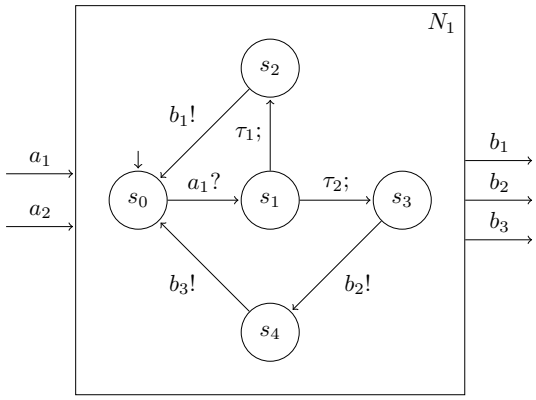


Figure 3: An example of a process $\mathcal{N}_1$ where LSIA $\overline{\mathcal{N}}_1$ describes the interaction protocol of $\mathcal{N}_1$.

Now, with a model to describe the interaction protocol of a process in a PNLSC we need to take a closer lock on how LSIAs interact with each other.

3.3 Interaction of LSIAs

We use LSIAs to describe the interaction protocol of processes in order to understand how processes interact with each other. The idea is to perform a binary composition of two LSIAs, each describing the protocol of a process, perform an analysis on the composition and then further compose it with another LSIA, describing another process of the PNLSC. This will be repeated until all processes of the PNLSC are included in the composition. In this section we describe how two LSIAs $\overline{\mathcal{M}}$ and $\overline{\mathcal{N}}$ interact with each other, and, in particular, how the actions of each particular LSIA are controlled.

Input and output ports of a process define the interface of the process to its environment. Input and output actions of a LSIA, describing the protocol of this process, define how the process behaviour is interacting with the environment. Shared actions, as defined by (6), represent the actual transmissions of messages from one process to another while shared ports represent the synchronous channels, spawned between processes. Open actions, as defined by(9), represent the ability of a process to communicate with its environment via ports where the corresponding communication partner is not (yet) known.

In order to understand how a LSIA reaches a state that is permanently blocking we have to understand

how actions are controlled in more detail. To do this we will first discuss the control of shared input and output actions. For this purpose let us consider the example in Figure 4. The two processes $\mathcal{M}_1$ and $\mathcal{N}_2$ share the ports a and b and their corresponding LSIAs $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_2$ share the action a. The output action b of LSIA $\overline{\mathcal{N}}_2$ is ignored.

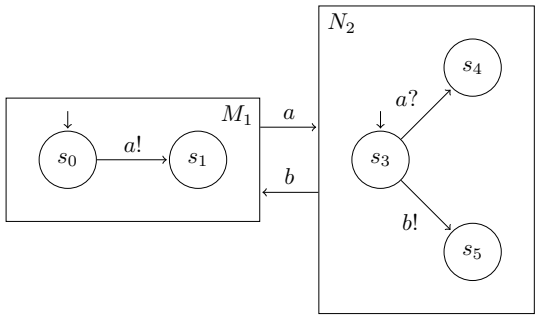


Figure 4: A simple example of two processes $\mathcal{M}_1$ and $\mathcal{N}_2$ with their corresponding LSIAs $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_2$ connected by the channels a and b. However, only action a is shared.

Initially, both, $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_2$, are in their initial states s_0 and s_3, respectively. From both of these states action a is enabled. Additionally, action b is enabled in state $s_3 \in S_{\overline{\mathcal{N}}_2}$. Hence, in state s_3 two transitions are possible. However, as the output action b will never be able to be served by $\mathcal{N}_2$'s environment (LSIA $\overline{\mathcal{M}}_1$ in this case) because no matching input action is available, the transition $\langle s_3, b, s_5\rangle$ will never be possible with the given environment $\overline{\mathcal{M}}_1$. Therefore, from their respective initial state s_0 and s_3, LSIAs $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_2$ transition to state s_1 and s_4, synchronized by the label a. The decision in state $s_3 \in S_{\overline{\mathcal{N}}_2}$ is imposed by the environment $\overline{\mathcal{M}}_1$.

Let us now study the control of open actions. Open actions are either of direction input or output, hence, blocking and non-autonomous. However, in contrast to shared actions, with open actions the environment is not (yet) known. Hence, an assumption has to be made whether the environment will eventually provide the corresponding counter parts of the open actions. The goal of describing the protocol behaviour of processes with the help of LSIA is to check all possible interaction behaviour in order to guarantee liveness. Hence, we will assume a *helpful* environment that is always providing open actions. This guarantees that potential permanent blocking states, that are only reachable through open actions, will still be considered. During the incremental composition of LSIAs, open actions will eventually become shared actions and will then be considered where the environment is known. Therefore, potential permanent blocking states caused by such actions will be detected at this later stage.

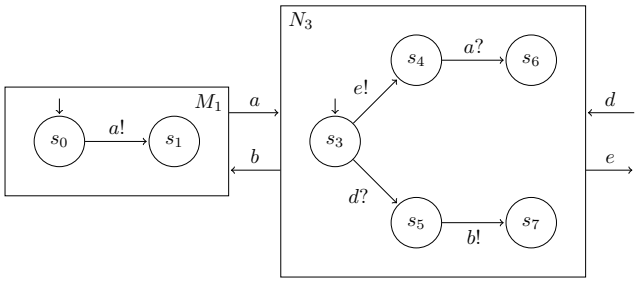


Figure 5: An example of a PNLSC where the LSIA $\overline{\mathcal{N}}_3$ of process $\mathcal{N}_3$ has the open actions d and e.

Note that open actions are, like shared actions, blocking and non-autonomous but because a helpful environment is assumed they will never block and will always be served. Hence, open actions will always trigger the corresponding transitions in a LSIA. Such an example is illustrated in Figure 5 where two processes $\mathcal{M}_1$ and $\mathcal{N}_3$ are connected by the shared ports a and b. The interaction protocol of process $\mathcal{M}_1$ is described by LSIA $\overline{\mathcal{M}}_1$. Process $\mathcal{N}_3$ has additional ports d and e which are not connected and its interaction protocol is described by LSIA $\overline{\mathcal{N}}_3$. LSIAs $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_3$ share the action a.

While LSIA $\overline{\mathcal{M}}_1$ resides in its initial state s_0, LSIA $\overline{\mathcal{N}}_3$ starts in state s_3 and can transition to either state s_4 or s_5 triggered by the open action e or d, respectively. As open actions are assumed to always be served by the environment, $\overline{\mathcal{N}}_3$ can reach either of the states s_4 and s_5 while LSIA $\overline{\mathcal{M}}_1$ is blocked in state s_0. With LSIA $\overline{\mathcal{N}}_3$ in state s_4, $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_3$ synchronise on the shared action a and transition to their respective state s_1 and s_6. Because the action $b \in \mathcal{A}_{\overline{\mathcal{N}}_3}$ is ignored, $\overline{\mathcal{N}}_3$ cannot perform any further transition from state s_5 to state s_7. Note that the decision at state $s_3 \in S_{\overline{\mathcal{N}}_3}$ depends on a currently unknown environment. At a later stage, a new process, interaction with its environment via ports d and e, might be added to the PNLSC. By this the system will gain knowledge of the environment, with respect to ports d and e and consequently, the corresponding actions of the respective LSIA will be turned into shared actions.

As established in the previous section, internal actions are controlled only by the process itself and not by the environment.

As established in the previous section, internal actions are controlled only by the process itself and not by the environment. Hence, internal actions are independent of interactions and do trigger transitions autonomously. The PNLSC illustrated in Figure 6 shows the two interacting processes M_1 and N_4. The inter-

action protocol of process $\overline{\mathcal{N}}_4$ includes transitions that are triggered by the internal actions τ_1 and τ_2.

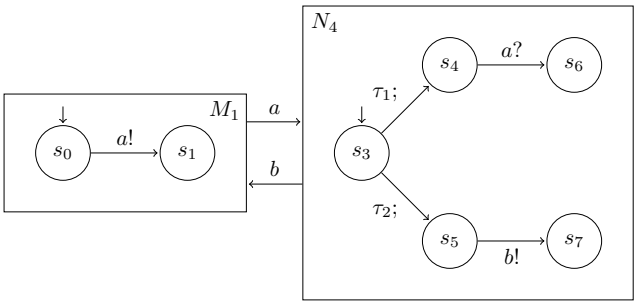


Figure 6: An example of a PNLSC where the LSIA $\overline{\mathcal{N}}_4$ of process $\mathcal{N}_4$ has the internal actions τ_1 and τ_2.

While LSIA $\overline{\mathcal{M}}_1$ temporarily blocks in its initial state s_0, LSIA $\overline{\mathcal{N}}_4$ starts in state s_3 and can transition to either state s_4 or s_5. The transitions are triggered by the internal actions τ_1 and τ_2, respectively. In contrast to the example of Figure 5 where open actions are triggering the transitions, here the transitions are independent of the environment. The choice at state $s_3 \in \mathcal{S}_{\overline{\mathcal{N}}_4}$ is unknown to the system because internal actions are hidden.

In Figure 6, the states $s_5 \in \mathcal{S}_{\overline{\mathcal{N}}_4}$ and $s_0 \in \mathcal{S}_{\overline{\mathcal{M}}_1}$ are permanent blocking states: With both LSIAs $\overline{\mathcal{M}}_1$ and $\overline{\mathcal{N}}_4$ starting in their respective initial state s_0 and s_3, an autonomous choice of process N_4 may lead to an internal transition of $\overline{\mathcal{N}}_4$ to state s_5. Because action $b \in \mathcal{A}^O_{\overline{\mathcal{N}}_4}$ is ignored, a further transition from state s_5 will never be possible. Hence, LSIA $\overline{\mathcal{N}}_4$ is blocking indefinitely in state s_5. At the same time LSIA $\overline{\mathcal{M}}_1$ is still blocking in state s_0 and waits for LSIA $\overline{\mathcal{N}}_4$ to reach state s_4 in order to synchronize on shared action a. In the current situation, this can never happen because LSIA $\overline{\mathcal{N}}_4$ is indefinitely blocking in state s_5 and therefore LSIA $\overline{\mathcal{M}}_1$ is indefinitely blocking in state s_0.

In contrast to the example with internal action (as depicted in Figure 6), the example with open actions (as depicted in Figure 5) has no permanent blocking states. This is because in Figure 5 the environment with respect to actions d and e might become known at a later stage and action d might be ignored, the transition $\langle s_3, d, s_5\rangle$ would not be possible and hence the permanent blocking state s_5 would never be reachable. Consequently, LSIA $\overline{\mathcal{M}}_1$ might never be permanently blocked in state s_0.

As an example of a composition, let us consider the PNLSC depicted in Figure 7 where two processes $\mathcal{M}_2$ and $\mathcal{N}_4$ share the ports a and b and process $\mathcal{M}_2$ has an unconnected port c. By folding their corresponding LSIAs $\overline{\mathcal{M}}_2$ and $\overline{\mathcal{N}}_4$ into the resulting LSIA $\overline{\mathcal{MN}}_1$, a

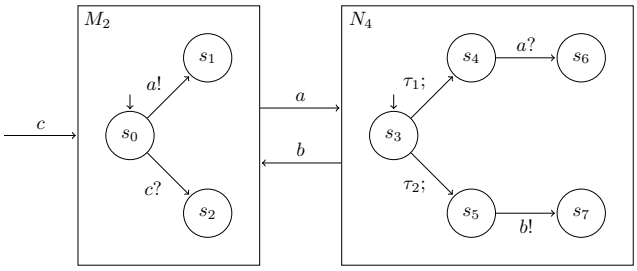


Figure 7: An example of a PNLSC with LSIAs containing open actions, shared actions, ignored actions and internal actions.

new composed process, we call it $\mathcal{MN}_1$ in this example, is created. The composed process $\mathcal{MN}_1$ with its corresponding LSIA $\overline{\mathcal{MN}}_1$ is depicted in Figure 8. All unreachable states of LSIA $\mathcal{MN}_1$ have been removed for the sake of readability.

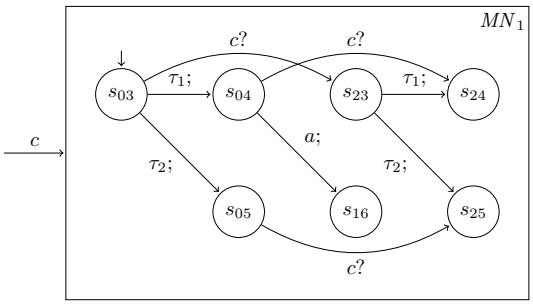


Figure 8: The composed process $\mathcal{MN}_1$ with its composed LSIA $\overline{\mathcal{MN}}_1$ as a result of the composition of the system depicted in Figure 7.

The only remaining port of $\mathcal{MN}_1$ is the unconnected port c which relates to the open action $c \in \mathcal{A}^I_{\overline{\mathcal{MN}}_1}$, triggering the transitions $\langle s_{03}, c, s_{23}\rangle$, $\langle s_{04}, c, s_{24}\rangle$, and $\langle s_{05}, c, s_{25}\rangle$. Note that all transitions triggered by action $c \in \mathcal{A}^I_{\overline{\mathcal{MN}}_1}$ change only the part of the state corresponding to the LSIA $\overline{\mathcal{M}}_2$. The reason for this is that it is not possible to reach s_{06} because $\overline{\mathcal{M}}_2$ must transition to state s_1 in order to synchronize on action a. State s_{07} cannot be reached because action $b \in \mathcal{A}^O_{\overline{\mathcal{N}}_4}$ is ignored. With similar reasoning, we identify the internal actions τ_1 and τ_2 that originate from LSIA $\overline{\mathcal{N}}_4$. The shared actions a of the LSIAs $\overline{\mathcal{M}}_2$ and $\overline{\mathcal{N}}_4$ are turned into the internal action $a \in \mathcal{A}^H_{\overline{\mathcal{MN}}_1}$, triggering the transition $\langle s_{04}, a, s_{16}\rangle$.

3.4 Modelling the Crossroad Example

In this section we will apply the LSIA model on the crossroad example described in Figure 2(a). We model the four crossroad sections by four processes

$\mathcal{P}_{NW}$, $\mathcal{P}_{NE}$, $\mathcal{P}_{SE}$, and $\mathcal{P}_{SW}$ together with their LSIAs $\overline{\mathcal{P}}_{NW}$, $\overline{\mathcal{P}}_{NE}$, $\overline{\mathcal{P}}_{SE}$, and $\overline{\mathcal{P}}_{SW}$, respectively, as depicted in Figure 9. The cars themselves are modelled as messages that arrive at one side of the crossroad and are transferred through the crossroad by subsequent messages.

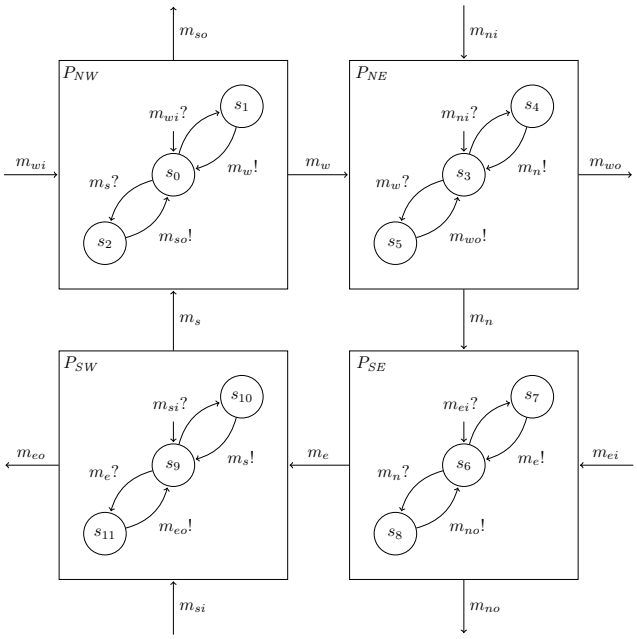


Figure 9: A PNLSC model of Figure 2(a) extended by the corresponding LSIAs of each process.

For example, a car arriving from the West going to the East is modelled by the messages sequence (m_{wi}, m_w, m_{wo}). Each LSIA in this example is modelled the same way, by non-deterministically accepting either a message of a car entering the crossroad section (e.g., m_{wi}) or a message of a car trying to transfer from inside the crossroad (e.g., m_s). What is important to mention here is that this model is a naive implementation with each crossroad section modelled symmetrically, which exhibits the deadlock as shown in Figure 2(b). To fix this deadlock, one would have to reverse the order of crossroad section allocations for one lane.

4 Conclusion and Future Work

In this paper we introduced *Loosely Synchronous Interface Automata* to model the interaction protocol of processes with their environment, based on loosely synchronous communication. Loosely synchronous communication is helpful for describing systems where a precise order of events is necessary. It includes cyber-physical systems that often contain control tasks where the exact order of events is necessary for a safe operation.

Our work is inspired by *Interface Automata* [3] and *Synchronous Interface Theories* [5], but it is fundamentally different in terms of the blocking semantics of the underlying models: PNLSCs with LSIAs can model loosely synchronous communication while the other models cannot.

As future work we plan to study how the model proposed in this paper can be used to verify harmonicity of interfaces.

References

[1] E. G. Coffman, M. Elphick, and A. Shoshani. System Deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971.

[2] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, and N. Yoshida. A gentle introduction to multiparty asynchronous session types. In M. Bernardo and E. B. Johnsen, editors, *Formal Methods for Multicore Programming - 15th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Advanced Lectures*, volume 9104 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2015.

[3] L. de Alfaro and T. A. Henzinger. Interface Automata. In *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, pages 109–120, New York, NY, USA, 2001. ACM.

[4] L. de Alfaro and T. A. Henzinger. Interface Theories for Component-Based Design. In T. A. Henzinger and C. M. Kirsch, editors, *Embedded Software*, number 2211 in Lecture Notes in Computer Science, pages 148–165. Springer Berlin Heidelberg, Oct. 2001.

[5] B. Delahaye, U. Fahrenberg, T. A. Henzinger, A. Legay, and D. Nickovic. Synchronous Interface Theories and Time Triggered Scheduling. In H. Giese and G. Rosu, editors, *Formal Techniques for Distributed Systems, International Conference, Proceedings*, volume 7273 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2012.

[6] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In G. C. Necula and P. Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 273–284. ACM, 2008.

[7] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2nd edition, 2011. ISBN: 978-1-4419-8236-0.

[8] S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee. A Theory of Synchronous Relational Interfaces. *ACM Transactions on Programming Languages and Systems*, 33(4):14:1–14:41, 2011.