On the Diversity of Asynchronous Communication

Florent Chevrou, Aurélie Hurault, Philippe Quéinnec

On the Diversity of Asynchronous Communication

Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec

IRIT – Université de Toulouse
ENSEEIHT
2 rue Camichel
F-31071 Toulouse, France
http://www.irit.fr

Abstract.

Asynchronous communication is often viewed as a single entity, the counterpart of synchronous communication. Although the basic concept of asynchronous communication is the decoupling of send and receive events, there is actually room for a variety of additional specification of the communication, for instance in terms of ordering. Yet, these different asynchronous communications are used interchangeably and seldom distinguished. This paper is a contribution to the study of these models, their differences, and how they are related.

In this paper, the variety of point-to-point asynchronous communication paradigms is considered with two approaches. In the first and theoretical one, communication models are specified as properties on the ordering of events in distributed executions. In the second and more practical approach that involves composition of peers, they are modeled with transition systems and message histories as part of a framework. The described framework enables to model peer composition and compatibility properties. Besides, an implemented tool chain based on the TLA$^+$ formalism and model checking is also proposed and illustrated.

The conformance of the two approaches is highlighted. A hierarchy is established between the studied communication models. From the execution viewpoint, it completes existing work in the area by introducing more asynchronous communication models and showing their differences. The framework is shown to offer abstract implementations of the communication models. Both the correctness and the completeness of the descriptions in the framework are studied. This reveals necessary restrictions on the behavior of the peers so that the communication models are actually implementable.

Keywords: asynchronous communication; formal verification; compatibility checking; TLA$^+$

Correspondence and offprint requests to: Philippe Quéinnec, e-mail: philippe.queinnec@irit.fr

1. Introduction

Building systems through selecting, and then assembling and coordinating off-the-shelf components or services is a common software production principle, which is emblematically illustrated by the development of Cloud-based services. The formal verification of the correctness of the composition of a set of peers is crucial to this approach. We consider this issue in the particular perspective of the development of distributed software systems taking into account the diversity of asynchronous communication models.

In this setting, the communication models (e.g. synchronous or asynchronous, multicast or point to point) directly impact the properties of the global system. Although the question of characterizing the properties of a set of combined services has been extensively studied for quite a long time (notions of design by contract [Mey92], of compatibility of communicating components [BZ83, LW11]), existing works are restricted, to the best of our knowledge, to a specific communication model (either synchronous or asynchronous, or coupling via bounded buffers), to which their formalization and verification framework are dedicated. However, in distributed algorithms research, it has long been known that the properties of the communication, and especially the order of message delivery, is essential to the correctness of the system. For instance, the Chandy-Lamport snapshot algorithm [CL85] requires that the communication between two processes is FIFO, and Misra's algorithm for termination detection [Mis83] works with a ring containing each *node* if the communication ensures causal delivery, but requires a cycle visiting all network *edges* if communication is only FIFO.

We present a framework to verify the correctness of a composition of peers with asynchronous communication. The first step is to explore the diversity of asynchronous communication models, to identify the more relevant ones, to uniformly define them, and to study the relations between them. Then, a framework is built to verify the correctness of a composition of peers parameterized by the communication model.

To get an intuition of the results, consider two peers (or processes, or services) described by the transition systems $\xrightarrow{a!} \cdot \xrightarrow{b!}$ and $\xrightarrow{a?} \cdot \xrightarrow{b?}$, where $a!$ and $b!$ are interpreted as emission events on channels a and b, and $a?$ and $b?$ are receive events on a and b. In the synchronous world (i.e. CCS), the compatibility of these two processes is well defined: both processes match on a first rendez-vous on a, then proceed to a second rendez-vous on b, and terminate. However, this is less clear in an asynchronous world. Traditionally, from a distributed systems point of view, one considers that the communication medium controls the message deliveries: it pushes messages up to the applications. Applications are limited to specify which channels they listen to, but they cannot impose a delivery order. In our example, if the communication medium ensures FIFO ordering (i.e. messages from one process to another are necessarily delivered in their emission order), then the message on a is delivered before the message on b, and we say that the two peers are compatible and terminate. However, if the communication medium is totally asynchronous and does not ensure any ordering, the message on b may be delivered before the message on a, but the second process does not expect this situation: compatibility is not guaranteed. Among the difficulties, a peer must be isolated from the other peers: it does not have to be ready for all kinds of messages. For instance, if the previous system also comprises two other peers $\xrightarrow{c?}$ and $\xrightarrow{c!}$, a message on c may be in transit. However, the communication medium will never deliver this message to the peer $\xrightarrow{a?} \cdot \xrightarrow{b?}$, as this message does not concern it. One last point is that the peers communicate through channels, and messages do not have an explicit destination process: our framework does not impose that channels have a unique sender and a unique receiver. This allows to naturally describe arbitrary client-server and publish-subscribe architectures.

The main results of this paper are:

- The unified description of seven asynchronous communication models (Section 2.4);
- The complete hierarchy of the models (Theorem 3.2);
- The formalization of the communication models (Table 2);
- The presentation of a framework to verify the correctness of a composition of peers with an asynchronous communication model (Definition 4.4);
- The proofs of correctness and completeness of the framework (Theorems 5.9 and 5.12).

The outline of this paper is the following. Section 2 presents the seven asynchronous communication models which are studied in this paper; the first part of this section recalls basic definitions and results of the theory of distributed systems, and then proceeds to present some standard and other less classic models. Section 3 compares these communication models with regard to executions, and gives their complete

hierarchy. Section 4 presents a framework which aims at checking compatibility properties over a composition of a set of peers and a communication model (possibly composite). Section 5 details the validation of the framework and shows the relations between the descriptions of the communication models in Section 2 and Section 4. Section 6 illustrates our approach with three case studies, and provides the results obtained with TLC, the TLA$^+$ model checker [Lam02]. Section 7 provides an overview of the conceptual background of this work and, eventually, the conclusion draws perspectives after summing up this work.

2. Communication Models: Concrete Definitions

This section introduces classic definitions of distributed executions and event ordering [BM93, Tel00, Ray13] and presents seven asynchronous point-to-point communication models in a unified way.

2.1. Distributed Systems

A message-passing distributed system is composed of a set of peers (or processes, or nodes) which exchange messages. Without loss of generality, messages are unique. A distributed execution is described with events: message send events, message receive events, and internal events (respectively noted $s(m)$, $r(m)$, and i). An execution is a sequence of events, obtained from the interleaving of peer sequences. As we only consider stable or local properties (termination, deadlock, faulty reception, never received message) and peers have no internal concurrency, interleaving and true parallelism are indistinguishable: the only properties which are not observable and not verifiable with interleaving are transitory global predicates [Cha97]. This paper only considers point-to-point communication, that is a message has exactly one sender and at most one receiver. Note that Definition 2.2 allows sent messages to never be received. This can be interpreted as the loss of the message, and is indistinguishable from a message staying forever in the network.

Definition 2.1 (Events). Let $\mathcal{PEER}$ be a set of peers, $\mathcal{MES}$ be an enumerable set of messages. The set of events is:

$$\mathcal{EVENT} \triangleq \{i_p \mid p \in \mathcal{PEER}\} \cup \{s_p(m), r_p(m) \mid p \in \mathcal{PEER}, m \in \mathcal{MES}\}$$

To ease notation, the index $_p$ is omitted and, when the peer is needed for an event e of an execution, $peer(e)$ is the peer where e occurs.

Definition 2.2 (Distributed Execution). $\mathcal{EXEC}$, the set of all possible executions over the events $\mathcal{EVENT}$, is the set of all finite or infinite sequences of events such that messages are unique, and such that a receive event of a message is preceded by a send event of this message:

$$\mathcal{SEQ} \triangleq \mathcal{EVENT}^* \cup \mathcal{EVENT}^\omega$$

$$\mathcal{EXEC} \triangleq \left\{ \sigma \in \mathcal{SEQ} \middle| \forall m \in \mathcal{MES} : \begin{array}{r} \forall j,k \in dom(\sigma) : \sigma[j] = s(m) \wedge \sigma[k] = s(m) \Rightarrow j = k \\ \wedge\ \forall j,k \in dom(\sigma) : \sigma[j] = r(m) \wedge \sigma[k] = r(m) \Rightarrow j = k \\ \wedge\ \forall j \in dom(\sigma) : \sigma[j] = r(m) \Rightarrow \exists k \in dom(\sigma) : \sigma[k] = s(m) \wedge k < j \end{array} \right\}$$

To simplify notation, we write $e \in \sigma$ to state that event e occurs in execution σ. It is a shortcut for $\exists j \in dom(\sigma) : \sigma[j] = e$.

Lemma 2.3 (Prefixes of an Execution). Any prefix of a distributed execution is a distributed execution.

Proof. From the definition of an execution. □

2.2. Event Ordering

When considering an execution σ, three orders are defined: a local order on each peer, the execution total order, and the causal order [Lam78] which abstracts independent events. The causal order relation is here given in its usual form [Lam78, Mat89, CDK94, Ray13].

Definition 2.4 (Execution Order). $e_1 \prec_\sigma e_2 \triangleq \exists j,k \in dom(\sigma) : j \leq k \wedge \sigma[j] = e_1 \wedge \sigma[k] = e_2$

M_{n-n}	M_{n-1}	M_{causal}	M_{1-1}	M_{1-n}
$s(m_1) \prec_\sigma s(m_2)$	$s(m_1) \prec_\sigma s(m_2) \wedge$ $peer(r(m_1)) = peer(r(m_2))$	$s(m_1) \prec_{causal} s(m_2) \wedge$ $peer(r(m_1)) = peer(r(m_2))$	$s(m_1) \prec_{peer} s(m_2) \wedge$ $peer(r(m_1)) = peer(r(m_2))$	$s(m_1) \prec_{peer} s(m_2)$

Table 1. Delivery Order Predicates

Definition 2.5 (Peer Order). $e_1 \prec_{peer} e_2 \triangleq peer(e_1) = peer(e_2) \wedge e_1 \prec_\sigma e_2$

Definition 2.6 (Causal Order). $\prec_{causal}$ is the smallest relation such that

$$\forall e_1, e_2 \in \sigma : e_1 \prec_{causal} e_2 \Leftrightarrow \begin{cases} e_1 \prec_{peer} e_2 & \text{(peer ordering)} \\ \vee\ \exists m \in \mathcal{MES} : e_1 = s(m) \wedge e_2 = r(m) & \text{(message transfer)} \\ \vee\ \exists e_3 \in \sigma : e_1 \prec_{causal} e_3 \wedge e_3 \prec_{causal} e_2 & \text{(transitivity)} \end{cases}$$

Theorem 2.7 (Order Inclusion).

$$\forall \sigma \in \mathcal{EXEC}, \forall e_1, e_2 \in \sigma : e_1 \prec_{peer} e_2 \Rightarrow e_1 \prec_{causal} e_2$$
$$\forall \sigma \in \mathcal{EXEC}, \forall e_1, e_2 \in \sigma : e_1 \prec_{causal} e_2 \Rightarrow e_1 \prec_\sigma e_2$$

Proof. This theorem is classic and directly derives from the definition of the orders and executions. □

2.3. Delivery Order

The communication enforces a delivery order if the order of message receptions is in relation with the order of their emissions. Thus, a communication model imposes an order on the message receptions, based on some specific predicate. An execution σ conforms to a communication model CM if all receptions are correctly ordered with regard to the delivery predicate of this model. Except for the first model (RSC), the set of valid executions of CM is defined according to a common pattern which takes the form:

$$\mathcal{E}xec(CM) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{l} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ \boxed{\text{condition on } s(m_1),\ s(m_2),\ r(m_1),\ r(m_2)} \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

where the delivery predicate is specific to the communication model and forces an order on two receive events based on some property on the send and receive events.

2.4. Asynchronous Communication Models

We present seven asynchronous point-to-point communication models, and five of them are summed up in Table 1 (fully asynchronous communication and RSC communication are omitted). There are four variants of FIFO communication, according to the peers involved: M_{1-1} coordinates one sender with one receiver, M_{n-1} coordinates all the senders of a unique receiver, M_{1-n} coordinates one sender with all its destinations, and M_{n-n} coordinates all the senders with all the receivers. Additionally, there are causal communication M_{causal}, pseudo-synchronous communication M_{rsc}, and fully asynchronous communication M_{async}. Applicative ordering, such as message priorities, could also be used. As our goal is to compare communication models without considering a specific application, such orderings are not considered.

2.4.1. M_{rsc} Realizable with Synchronous Communication

An execution is *realizable with synchronous communication* if each send event is immediately followed by its corresponding receive event [CBMT96, KS11]. If the couple (send event, corresponding receive event) is viewed atomically, this corresponds to a synchronous communication execution. Figure 1 illustrates M_{rsc}.

Definition 2.8 (RSC).

$$\mathcal{E}xec(M_{rsc}) \triangleq \{\sigma \in \mathcal{EXEC} \mid \forall m \in \mathcal{MES} : r(m) \in \sigma \Rightarrow \forall e \in \sigma : s(m) \prec_\sigma e \prec_\sigma r(m) \Rightarrow e \in \{r(m), s(m), i\}\}$$

This model can be implemented with a 1-slot unique buffer shared by all peers. After a communication transition consisting of a send event of a message, the only possible communication transition is the receive

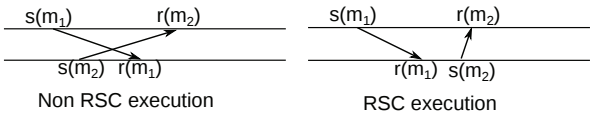


Fig. 1. RSC executions

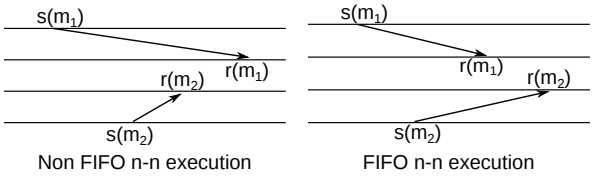


Fig. 2. FIFO n-n executions

event of this message. Thus, this asynchronous model is the closest one to synchronous communication. The difference is that this model allows interleaving of internal transitions (by the sender, the receiver, or any other peer) between the sending of a message and its reception, and can avoid deadlock by divergence with one never received message (an execution $s(m)\ i^{\omega}$).

2.4.2. M_{n-n} FIFO n-n Communication

Messages are globally ordered and are delivered in their emission order. This model is based on a shared centralized object (e.g. a unique queue), and its implementations are inefficient and unrealistic. Usually, it is used as a first step to move away from the synchronous communication model, by splitting send and receive events. Figure 2 illustrates M_{n-n}.

Definition 2.9 (FIFO n-n).

$$\mathcal{E}xec(M_{n-n}) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{r} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ s(m_1) \prec_\sigma s(m_2) \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

A variant exists where the number of messages in transit is bounded. With $prefix(\sigma, j)$ being the prefix of an execution σ up to point j, this can be expressed as:
$\max_{j \in dom(\sigma)} |\{m \in \mathcal{MES} : s(m) \in prefix(\sigma, j) \wedge r(m) \notin prefix(\sigma, j)\}| \leq bound.$

The other models have similar bounded variants which are not detailed.

2.4.3. M_{n-1} FIFO n-1 Communication

Each peer has a unique input queue (a.k.a mailbox). A send event consists in adding the message at the end of the queue of the destination peer, without blocking. The message will later be removed from the queue, according to the insertion order. This model is used for instance in [BBO12, OSB13] as an abstraction of asynchronous communication. This model is often confused with the FIFO 1-1 model (described below) whereas it is stricter. Even if the send events are independent, the delivery order is their send order in absolute time. A send event is implicitly and globally ordered with regard to all other emissions toward the same peer. This means that if a peer p consumes m_1 (sent by a peer q_1) and later m_2 (sent by peer q_2), peer p *knows* that the sending on peer q_1 occurs before the sending on peer q_2 in the global execution order, even if there is no causal path between the two emissions. Thus, an implementation of this model requires a shared real-time clock [CF99] or a global agreement on event order [DSU04, Ray10]. Figure 3 illustrates M_{n-1}.

Definition 2.10 (FIFO n-1).

$$\mathcal{E}xec(M_{n-1}) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{r} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ s(m_1) \prec_\sigma s(m_2) \wedge peer(r(m_1)) = peer(r(m_2)) \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

2.4.4. M_{1-n} FIFO 1-n Communication

Messages from a same peer are delivered in their send order. This model is the dual of FIFO n-1, but is less intuitive as it is inducing a separate global order on the receivers, one for each sender. Its implementation is not expensive: each peer has a unique queue where sent messages are put. Destination peers fetch messages from this queue and acknowledge their reception. Figure 4 illustrates M_{1-n}.

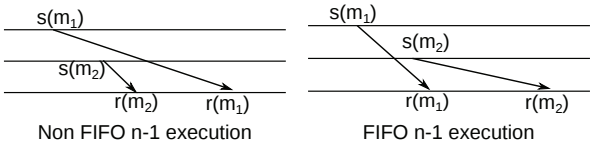


Fig. 3. FIFO n-1 executions

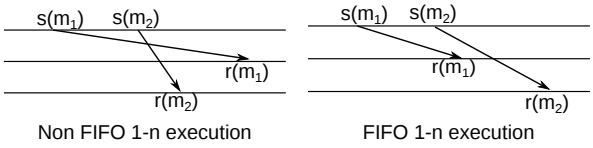


Fig. 4. FIFO 1-n executions

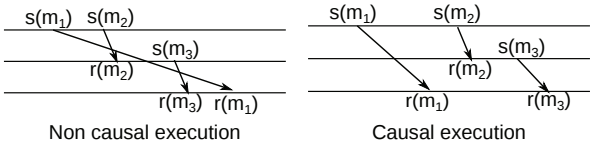


Fig. 5. Causal executions

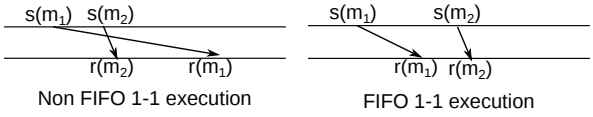


Fig. 6. FIFO 1-1 executions

Definition 2.11 (FIFO 1-n).

$$\mathcal{E}xec(M_{1-n}) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{r} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ s(m_1) \prec_{peer} s(m_2) \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

(the form of the delivery predicate ensures that the send events occur on the same peer)

2.4.5. M_{causal} Causally Ordered Communication

Messages are delivered according to the causality of their emissions [Lam78]. More precisely, if a message m_1 is causally sent before a message m_2 (i.e. there exists a causal path from the first emission to the second one), then a peer cannot get m_2 before m_1. An implementation of this model requires the sharing of the causality relation, using causal histories [SM94, KS98] or logical vector/matrix clocks [RST91, CDK94, PRS97, Ray13]. Figure 5 illustrates M_{causal}.

Definition 2.12 (Causal).

$$\mathcal{E}xec(M_{causal}) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{r} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ s(m_1) \prec_{causal} s(m_2) \wedge peer(r(m_1)) = peer(r(m_2)) \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

2.4.6. M_{1-1} FIFO 1-1 Communication

Messages between a couple of peers are delivered in their send order. Messages from/to different peers are independently delivered. More precisely, if a peer sends a message m_1 and later a message m_2, and these two messages are consumed by a same peer, then m_2 cannot be consumed before m_1. This model is easily described with a simple queue between each pair of peers. An implementation can be as inexpensive as an integer counter for each pair of peers. Figure 6 illustrates M_{1-1}.

Definition 2.13 (FIFO 1-1).

$$\mathcal{E}xec(M_{1-1}) \triangleq \left\{ \sigma \in \mathcal{EXEC} \;\middle|\; \forall m_1, m_2 \in \mathcal{MES} : \begin{array}{r} r(m_1) \in \sigma \wedge r(m_2) \in \sigma \\ \wedge\ s(m_1) \prec_{peer} s(m_2) \wedge peer(r(m_1)) = peer(r(m_2)) \\ \Rightarrow r(m_1) \prec_\sigma r(m_2) \end{array} \right\}$$

2.4.7. M_{async} Fully Asynchronous Communication

No order on message delivery is imposed. Messages can overtake others or be arbitrarily delayed. The implementation is usually modeled by a bag (or a set if messages are unique).

Definition 2.14 (Asynchronous).

$$\mathcal{E}xec(M_{async}) \triangleq \mathcal{EXEC}$$

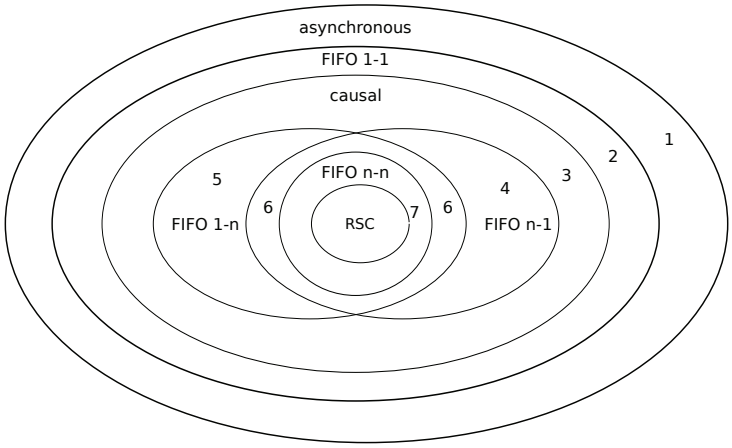


Fig. 7. Hierarchy of the Communication Models

2.4.8. Execution Prefixes

Theorem 2.15.

$$\forall M \in \{M_{rsc}, M_{n-n}, M_{n-1}, M_{causal}, M_{1-n}, M_{1-1}, M_{async}\}, \forall \sigma \in \mathcal{EXEC} :$$
$$\sigma \in \mathcal{E}xec(M) \Leftrightarrow \forall \sigma' \in \mathit{prefixes}(\sigma) : \sigma' \in \mathcal{E}xec(M)$$

Proof. All delivery conditions are exclusively safety properties. Thus, all prefixes of an execution which satisfies a delivery condition satisfy this delivery condition. And conversely, if all finite prefixes satisfy it, the (possibly infinite) execution satisfies it. □

3. Comparison of the Communication Models

3.1. Hierarchy of the Models

The executions which are valid for a given model may also be valid for another model. This means that, for these executions, we can switch whatever of these models is considered. Knowing which model can be replaced by which is of great interest to study the potential compatibility of peers. Model inclusions are given by Theorem 3.2, which is illustrated in Figure 7.

Definition 3.1 (Model Inclusion). A communication model M_1 is included in M_2 iff all valid executions of M_1 are also valid in M_2:

$$M_1 \subseteq M_2 \triangleq \mathcal{E}xec(M_1) \subseteq \mathcal{E}xec(M_2)$$

Theorem 3.2 (Hierarchy of the Communication Models).

- $M_{rsc} \subsetneq M_{n-n} \subsetneq M_{n-1} \subsetneq M_{causal} \subsetneq M_{1-1} \subsetneq M_{async}$
- $M_{rsc} \subsetneq M_{n-n} \subsetneq M_{1-n} \subsetneq M_{causal} \subsetneq M_{1-1} \subsetneq M_{async}$
- M_{1-n} and M_{n-1} are not comparable.

3.2. Proof of Theorem 3.2

3.2.1. Simple Proofs

Almost all inclusions directly derive from the delivery predicates and the order inclusion (Theorem 2.7).

- $M_{rsc} \subseteq M_{n-n}$: M_{rsc} is M_{n-n} constrained by a bound of 1.

- $M_{n-n} \subseteq M_{n-1}$: The delivery predicate of M_{n-1} implies the delivery predicate of M_{n-n}, thus $\forall \sigma \in \mathcal{E}xec(M_{n-n}) : \sigma \in \mathcal{E}xec(M_{n-1})$.
- $M_{n-1} \subseteq M_{causal}$: From Theorem 2.7, the delivery predicate of M_{causal} implies the delivery predicate of M_{n-1}, thus $\forall \sigma \in \mathcal{E}xec(M_{n-1}) : \sigma \in \mathcal{E}xec(M_{causal})$.
- $M_{causal} \subseteq M_{1-1}$: From Theorem 2.7, the delivery predicate of M_{1-1} implies the delivery predicate of M_{causal}, thus $\forall \sigma \in \mathcal{E}xec(M_{causal}) : \sigma \in \mathcal{E}xec(M_{1-1})$.
- $M_{n-n} \subseteq M_{1-n}$: From Theorem 2.7, the delivery predicate of M_{1-n} implies the delivery predicate of M_{n-n}, thus $\forall \sigma \in \mathcal{E}xec(M_{n-n}) : \sigma \in \mathcal{E}xec(M_{1-n})$.

3.2.2. Proof of $M_{1-n} \subseteq M_{causal}$

To facilitate induction, the causality relation (Definition 2.6) is rewritten by expanding the base case in the transitivity formula:

Lemma 3.3 (Causal Order). $\prec_{causal}$ is the smallest relation on σ such that $\forall e_1, e_2 \in \sigma$:

$$e_1 \prec_{causal} e_2 \Leftrightarrow \begin{cases} e_1 \prec_{elem} e_2 \\ \text{or } \exists e_3 \in \sigma, e_1 \neq e_3 : e_1 \prec_{elem} e_3 \wedge e_3 \prec_{causal} e_2 \end{cases}$$

$$\text{where } e_1 \prec_{elem} e_2 \triangleq \begin{cases} e_1 \prec_{peer} e_2 \\ \text{or } \exists m \in \mathcal{MES}, e_1 = s(m) \wedge e_2 = r(m) \end{cases}$$

Proof. This is the transformation of "transitivity n-n" to "transitivity 1-n", and the two definitions are proven equivalent in Coq Standard Library (module Relations). □

Lemma 3.4 (Causal events on different peers are linked by a message).

$$\forall \sigma \in \mathcal{EXEC}, \forall e_1, e_2 \in \sigma : e_1 \prec_{causal} e_2 \wedge peer(e_1) \neq peer(e_2) \Rightarrow \exists m : e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$$

This lemma states that it is impossible to have two causally related events on different peers without at least one message between them: messages are the only causal links between peers.

Proof. Let $\sigma \in \mathcal{EXEC}$ an execution, and e_1, e_2 such that $e_1, e_2 \in \sigma$, $e_1 \prec_{causal} e_2$ and $peer(e_1) \neq peer(e_2)$. Let's prove that $\exists m, e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$ by induction on the principle underlying Lemma 3.3.

- Case $e_1 \prec_{elem} e_2$: Either $e_1 \prec_{peer} e_2$, which is incompatible with the hypothesis $peer(e_1) \neq peer(e_2)$. Or $\exists m, e_1 = s(m) \wedge e_2 = r(m)$. So $\exists m, e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$ (since $\prec_{peer}$ and $\prec_{causal}$ are order relations, they are reflexive).
- Case $\exists e_3 \in \sigma : e_3 \neq e_1 \wedge e_1 \prec_{elem} e_3 \wedge e_3 \prec_{causal} e_2$, with the induction hypothesis true for $e_3 \prec_{causal} e_2$:
  - either $peer(e_1) = peer(e_3)$, so $peer(e_3) \neq peer(e_2)$. By the induction hypothesis: $\exists m, e_3 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$. Since $e_1 \prec_{peer} e_3$, the transitivity of the peer order gives $\exists m, e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$.
  - or $peer(e_1) \neq peer(e_3)$. From Lemma 3.3, it follows that $\exists m, e_1 = s(m) \wedge e_3 = r(m)$, so $\exists m, e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_3$. The transitivity of the causal order gives $\exists m, e_1 \prec_{peer} s(m) \wedge r(m) \prec_{causal} e_2$.

□

Lemma 3.5 (Reception of causal send events in a valid FIFO 1-n execution).

$$\forall \sigma \in \mathcal{E}xec(M_{1-n}), \forall m_1, m_2 \in \mathcal{MES} : s(m_1) \prec_{causal} s(m_2) \wedge r(m_1) \in \sigma \wedge r(m_2) \in \sigma \Rightarrow r(m_1) \prec_{\sigma} r(m_2)$$

This lemma expresses that, in the M_{1-n} model, the reception of a message m_2 ensures that a causally preceding message m_1 cannot be received later.

Proof.[1] Let $\sigma \in \mathcal{E}xec(M_{1-n})$, and m_1, m_2 such that $s(m_1) \prec_{causal} s(m_2)$. Let's prove that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \Rightarrow r(m_1) \prec_{\sigma} r(m_2)$.

- either $peer(s(m_1)) = peer(s(m_2))$

[1] In the next proofs, we use $X \Longrightarrow_{(Thm.x)} Y$ to mean "from X, Theorem x (or definition or lemma...), *and the preceding properties/hypotheses*, we deduce that Y".

  - $s(m_1) \prec_{causal} s(m_2) \Longrightarrow_{(Thm.\ 2.7)} s(m_1) \prec_\sigma s(m_2)$
  - $peer(s(m_1)) = peer(s(m_2)) \Longrightarrow_{(Def.\ 2.5, Thm.\ 2.7)} s(m_1) \prec_{peer} s(m_2) \Longrightarrow_{(Def.\ 2.11)} (r(m_1) \in \sigma \wedge r(m_2) \in \sigma \Rightarrow r(m_1) \prec_\sigma r(m_2))$.
- or $peer(s(m_1)) \neq peer(s(m_2))$
  - $peer(s(m_1)) \neq peer(s(m_2)) \Longrightarrow_{(Lemma\ 3.4)} \exists m, s(m_1) \prec_{peer} s(m) \wedge r(m) \prec_{causal} s(m_2)$, so $r(m) \in \sigma$.
  - $\sigma \in \mathcal{E}xec(M_{1-n}) \wedge r(m) \in \sigma \wedge s(m_1) \prec_{peer} s(m) \Longrightarrow_{(Def.\ 2.11)} (r(m_1) \in \sigma \Rightarrow r(m_1) \prec_\sigma r(m))$
  - $r(m_2) \in \sigma \Longrightarrow_{(Def.\ 2.2)} s(m_2) \prec_\sigma r(m_2)$
  - $r(m) \prec_{causal} s(m_2) \wedge s(m_2) \prec_\sigma r(m_2) \Longrightarrow_{(Thm.\ 2.7,\ transitivity\ of\ \prec_\sigma)} r(m) \prec_\sigma r(m_2)$
  - so $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \Rightarrow r(m_1) \prec_\sigma r(m_2)$

□

From Lemma 3.5, we deduce that $\forall \sigma \in \mathcal{E}xec(M_{1-n}) : \sigma \in \mathcal{E}xec(M_{causal})$.
This concludes the inclusion proofs of Theorem 3.2.

3.2.3. Proofs of Strict Inclusion

All inclusions are strict, and M_{1-n} and M_{n-1} are not comparable. It suffices to give an example in each case, numbered 1 to 7 in Figure 7. Examples are a system composed of a set of peers (described by a labelled transition system), and an execution which is valid in a communication model, invalid in another model.

	System	Execution
1 asynchronous, not FIFO 1-1	$\xrightarrow{s(a)}\xrightarrow{s(b)} \parallel \xrightarrow{r(b)}\xrightarrow{r(a)}$	$s(a)\ s(b)\ r(b)\ r(a)$
2 FIFO 1-1, not causal	$\xrightarrow{s(a)}\xrightarrow{s(b)} \parallel \xrightarrow{r(b)}\xrightarrow{s(c)} \parallel \xrightarrow{r(c)}\xrightarrow{r(a)}$	$s(a)\ s(b)\ r(b)\ s(c)\ r(c)\ r(a)$
3 Causal, neither n-1 nor 1-n	$\xrightarrow{s(a)}\xrightarrow{s(b)} \parallel \xrightarrow{s(c)} \parallel \xrightarrow{r(c)}\xrightarrow{r(a)} \parallel \xrightarrow{r(b)}$	$s(a)\ s(b)\ s(c)\ r(c)\ r(b)\ r(a)$
4 n-1, neither 1-n nor n-n	$\xrightarrow{s(a)}\xrightarrow{s(b)} \parallel \xrightarrow{r(a)} \parallel \xrightarrow{r(b)}$	$s(a)\ s(b)\ r(b)\ r(a)$
5 1-n, neither n-1 nor n-n	$\xrightarrow{s(a)} \parallel \xrightarrow{s(b)} \parallel \xrightarrow{r(a)}\xrightarrow{r(b)}$	$s(b)\ s(a)\ r(a)\ r(b)$
6 1-n and n-1, not n-n	$\xrightarrow{s(a)} \parallel \xrightarrow{s(b)} \parallel \xrightarrow{r(a)} \parallel \xrightarrow{r(b)}$	$s(a)\ s(b)\ r(b)\ r(a)$
7 n-n, not RSC	$\xrightarrow{s(a)}\xrightarrow{s(b)} \parallel \xrightarrow{r(a)}\xrightarrow{r(b)}$	$s(a)\ s(b)\ r(a)\ r(b)$

4. A Framework for the Verification of Asynchronously Communicating Peers [2]

Our objective is to check the compatibility of the composition of a set of peers, given a behavorial description of the peers and a communication model. One solution could be to extensionally build all the executions of this set of peers, then reduce this set by keeping only the relevant executions of the chosen communication model, and lastly evaluate properties on this set (non-emptiness, temporal properties...). This solution would strictly reuse the predicates of Section 2. However, it is not satisfactory because the predicates are about whole executions and are not directly implementable: they are oracles, in the sense that a reception is valid at a given point with regard to all events, before and *after* it. For instance, consider M_{1-1} and two messages m_1 and m_2 sent by the same peer in this order; the reception of m_2 on a peer p at time t is correct if this peer p will never receive m_1 at time $t' > t$. The verification of compatibility has to be of practical use: the decision by a communication model to deliver a message must be based only on past events, and not on the possible occurrence of future events. Our description of the communication models should be operational but it should also be as abstract as possible, to not preclude implementations, so that the compatibility can be ascertained for any realistic implementations. The formalization is based on histories which grab just enough information on the past to build valid executions. The framework is presented in two sections: this section describes the framework itself, and the next section demonstrates its validity. Its correctness (the executions generated by the framework for a communication model are correct)

[2] This section and the example section 6 are expanded versions of [CHQ15]. The other sections were not part of it.

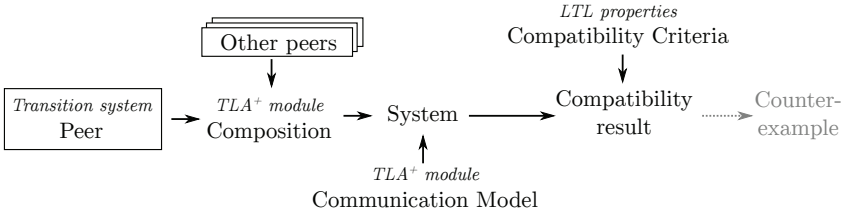


Fig. 8. Main Steps Performed by the Framework

and completeness (the framework generates all possible executions for a model) require well-behaved peers, so that the communication model does not have to look at the future to make a correct decision.

Communication consists in exchanging messages whose content is not relevant outside the scope of peers' internal behavior. Messages are sent on channels. Channels do not have explicit sender and receiver, and are not limited to one sender/one receiver. Channels are nevertheless a point-to-point communication abstraction: a given message has exactly one sender and at most one receiver. This allows for richer and more elegant system specifications, where for instance a message is received by a peer that depends on the state of the communication medium.

Peers are specified using transition systems labelled by communication events or internal actions. This provides simplicity and flexibility in theoretical and practical uses. It is a basic structure that is adapted to verification techniques like model checking. It also allows to specify properties of interest using temporal logic. Optionally, in order to ease the specification of peers, the transitions systems can be derived from process calculi terms such as CCS [Mil82]. Since communication models are specified using similar transition systems, modeling the communication interactions simply consists in an operation close to a synchronous product of labeled transition systems.

All these notions are directly translated into TLA$^+$ specifications, and the TLA$^+$ tools are used to verify the compatibility properties. Figure 8 provides an overview of the different steps and elements used to perform the verification of a composition. They will be detailed in the following sections.

4.1. TLA$^+$ Specification Language

TLA$^+$ [Lam02] is a formal specification language based on untyped Zermelo-Fraenkel set theory for specifying data structures, and on the temporal logic of actions (TLA) for specifying dynamic behaviors. Expressions rely on standard first-order logic, set operators, and several arithmetic modules. Hilbert's choice operator, written as CHOOSE $x \in S : p$, deterministically picks an arbitrary value in S which satisfies p, provided such a value exists (its value is undefined otherwise).

Functions are primitive objects in TLA$^+$, and tuples are a particular kind of function. The application of function f to an expression e is written as $f[e]$. The set of functions whose domain is X and whose co-domain is a subset of Y is written as $[X \to Y]$. The expression DOMAIN f is the domain of the function f. The expression $[x \in X \mapsto e]$ denotes the function with domain X that maps any $x \in X$ to e. The notation $[f$ EXCEPT $![e_1] = e_2]$ is a function which is equal to the function f except at point e_1, where its value is e_2. Tuples (a.k.a sequences) are functions with domain $1..n, n \in Nat$. Tuples are written $\langle a_1, a_2, a_3 \rangle$. $\langle\rangle$ is the empty sequence.

Modules are used to structure complex specifications. A module contains constant declarations, variable declarations, and definitions. A module can *extend* other modules, importing all their declarations and definitions. A module can also be an *instantiation* of another module. The module $MI \triangleq$ INSTANCE M WITH $q_1 \leftarrow e_1, q_2 \leftarrow e_2 \ldots$ is an instantiation of module M, where each symbol q_i is replaced by e_i (q_i are identifiers specifying constants or variables of module M, and e_i are expressions). Then $MI!x$ references the symbol x of the instantiated module.

Other than constant and variable declarations, a module contains definitions in the form $Op(arg_1, \ldots, arg_n) \triangleq exp$. This defines the symbol Op such that $Op(e_1, \ldots, e_n)$ equals exp, where each arg_i is replaced by e_i. In case of no argument, it is written as $Op \triangleq e$. A definition is just an abbreviation or syntactic sugar for an expression, and never changes its meaning.

The dynamic behavior of a system is expressed in TLA$^+$ as a transition system, with an initial state predicate, and *actions* to describe the transitions. An action formula describes the changes of state variables

after a transition. In an action formula, x denotes the value of a variable x in the origin state, and x' denotes its value in the destination state. A prime is never used to distinguish symbols but always means "in the next state". ENABLED A is a predicate which is true in a state iff the action A is feasible, i.e. there exists a next state such that A is true.

A specification of a system is written as $\mathit{Init} \wedge \Box[\mathit{Next}]_{vars} \wedge \mathcal{F}$, where Init is a predicate specifying the initial states, $\Box$ is the temporal operator which asserts that the formula following it is always true, Next is the transition relation, usually expressed as a disjunction of actions, $[\mathit{Next}]_{vars}$ is defined to equal $\mathit{Next} \vee vars' = vars$ (Next with stuttering), and $\mathcal{F}$ expresses fairness conditions. Fairness is usually expressed as a conjunction of weak or strong fairness on actions $\mathrm{WF}_{vars}(A_1) \wedge \mathrm{WF}_{vars}(A_2) \ldots \wedge \mathrm{SF}_{vars}(A_i) \ldots$. Weak fairness $\mathrm{WF}_v(A)$ means that either infinitely many A steps occur or A is infinitely often disabled. In other words, an A step must eventually occur if A is continuously enabled. Strong fairness $\mathrm{SF}_v(A)$ means that either infinitely many A steps occur or A is eventually disabled forever. In other words, an A step must eventually occur if A is repeatedly enabled.

System properties are specified using linear temporal logic (LTL). $\Box\phi$ means that ϕ holds in every suffix of the behavior. $\Diamond\phi$ is defined to equal $\neg\Box\neg\phi$ and means that ϕ eventually holds in a subsequent state. $\psi \rightsquigarrow \phi$ is defined to equal to $\Box(\psi \Rightarrow \Diamond\phi)$ and means that, whenever ψ holds, then later ϕ holds.

4.2. System Model

Definition 4.1 (Peer). Let $\mathcal{C}$ be an enumerable set of channels. A peer $\mathcal{P}_p$ is a labeled transition system $TS_p = (S_p, I_p, R_p, L_p)$ where S_p is the set of states, I_p is the set of initial states ($I_p \subseteq S_p$), L_p is a (enumerable) set of labels, and $R_p \subseteq S_p \times L_p \times S_p$ is the transition relation.

The set of labels L_p contains τ and a subset of $\bigcup_{c \in \mathcal{C}} \{c!, c?\}$. τ is an internal action and we assume stuttering: $\forall s \in S_p : s \xrightarrow{\tau} s \in R_p$. The labels $c!$ and $c?$ are interpreted as the sending of a message on channel c, and the reception of a message from channel c.

To describe communication properties, we need to know the listened channels in a given state. For instance, consider a state s where messages on the channels c_1 and c_2 may be handled by $\mathcal{P}_p$ (this means that $\exists s_1 \in S_p : s \xrightarrow{c_1?} s_1 \in R_p$ and $\exists s_2 \in S_p : s \xrightarrow{c_2?} s_2 \in R_p$), and two messages are in transit on c_1 and c_2. If the communication ensures a FIFO ordering and both messages have the same sender, then only the oldest one may be delivered to the peer. Thus, the other transition must be disabled in this configuration. However, if in the state s only the channel c_2 is listened to, the message on c_2 may be delivered even if it is younger than the message on c_1. Recall that it is the communication model which pushes messages to the peer.

Definition 4.2 (Listened Channels). Let s be a state in S_p,

$$LC_p(s) \triangleq \{c \in \mathcal{C} \mid \exists s_1 \in S_p : s \xrightarrow{c?} s_1 \in R_p\}$$

Definition 4.3 (Communication Model). A communication model $\mathcal{CM}$ is a labelled transition system with stuttering $(S_{CM}, I_{CM}, R_{CM}, L_{CM})$, where S_{CM}, I_{CM}, R_{CM} and L_{CM} have the same meaning as above (states, initial states, transition relation, labels).

The set of labels contains τ, a subset of $\mathbb{N} \times \bigcup_{c \in \mathcal{C}} \{c!\}$ (send events by peer p on channel c), and a subset of $\mathbb{N} \times \bigcup_{c \in \mathcal{C}} \{c?\} \times \mathcal{P}(\mathcal{C})$ (receive events by peer p on channel c while listening to a set of channels).

The actual definition of a communication model depends on its characteristics and examples are provided in Section 4.3.

Definition 4.4 (Composed System). The composed system $\mathit{System} = (S, I, R)$ is the product of the $TS_p : p \in 1..N$ with a communication model $\mathcal{CM}$

- $S = S_1 \times \ldots \times S_N \times S_{CM}$
- $I = I_1 \times \ldots \times I_N \times I_{CM}$
- $$R = \left\{ s \to s' \;\middle|\; \left(\begin{array}{c} \textbf{Internal actions} \\ s_{cm} \xrightarrow{\tau} s'_{cm} \in R_{cm} \\ \wedge \quad \forall p \in 1..N : s_p \xrightarrow{\tau} s'_p \in R_p \end{array} \right) \vee \left(\begin{array}{c} \textbf{Communication} \\ \exists p \in 1..N : \exists c \in \mathcal{C} : \\ \mathrm{send}(s, s', p, c) \\ \vee \quad \mathrm{receive}(s, s', p, c) \end{array} \right) \right\}$$

Thus, a system state s is a tuple $(s_1, \ldots, s_N, s_{cm})$. Given a system state s, we note s_p the projection $\pi_p(s)$ of s on $\mathcal{P}_p$, and s_{cm} the projection of s on $\mathcal{CM}$.

The transition relation of the composed system contains on the one hand the internal actions of each peer, and on the other hand communication transitions. Contrary to the synchronous model, where communication is a rendez-vous between two peers, asynchronous communication is modeled with distinct send actions and receive actions:

$$\text{send}(s, s', p, c) \triangleq \begin{cases} s_p \xrightarrow{c!} s'_p \in R_p \\ s_{cm} \xrightarrow{p,c!} s'_{cm} \in R_{cm} \\ s_k \xrightarrow{\tau} s'_k \in R_k, \forall k \neq p, cm \end{cases}$$

$$\text{receive}(s, s', p, c) \triangleq \begin{cases} s_p \xrightarrow{c?} s'_p \in R_p \\ s_{cm} \xrightarrow{p,c?,L} s'_{cm} \in R_{cm} \text{ where } L = LC_p(s_p) \\ s_k \xrightarrow{\tau} s'_k \in R_k, \forall k \neq p, cm \end{cases}$$

Internal actions allow for parallel internal evolution of the peers and the communication model, but it does not allow for parallel independent communication actions. However, as stated in Section 2.1, interleaving and parallelism are equivalent as the compatibility properties in 4.4 are all stable properties. To avoid infinite stuttering, we assume a minimal progress property: the peers and the communication model can infinitely stutter only if no other transition can be done. In the TLA$^+$ specification, this is obtained by having weak fairness (WF) on every action.

Figure 10 (see Figure 9 for the *trans* action in the *peermanagement* module) shows an example of a system composed of two peers. The module instantiates the *causal* communication model to get the send and receive actions. The two peers are respectively initialized in states 11 and 14. Two transitions depart from state 14, depending on the reception channel. In the following, the terminal states are differentiated between correct terminal states and faulty states reached because of an unexpected reception (for instance, state 17 is unreachable with the causal communication model, but is reachable with the fully asynchronous model). Details are provided in the following sections, especially in 4.4 and 4.5.2.

4.3. Asynchronous Communication Models

The seven asynchronous communication models are described in Table 2. This table contains the logical descriptions of the communication models, not their implementations. An actual implementation of the system would use more efficient realizations of the communication models, such as vector/matrix clocks for causality, or numbering for FIFO ordering.

In the description, all communication models have a *net* variable, initially empty, which holds the messages in transit. A message is ⟨ channel, sender, message history ⟩, which consists of a channel, an emitting site identifier, and a history of previously sent messages. Depending on the communication model, a global history variable H or a tuple of history variables H_p (one for each peer), also initially empty, are used[3]. Histories hold the set of all sent messages, or the set of sent messages by a peer, or the current causal past (the set of messages on which the next emission is causally dependent). The history variables have two roles:

- Carrying the logical information in the distributed system to ensure ordering properties. Typically, a reception predicate can require that no message in transit appears in the history of the to-be-received message. The history and possibly the sender are used to decide on the delivery of the message (receive action).
- Identifying messages: history variables always expand by appending new messages in which their previous value appear. Using histories provides a way to ensure the uniqueness of messages.

As an example, Figure 11 shows the TLA$^+$ module corresponding to the causal communication model. It is a direct translation of the formulae of Table 2. All the models are available at `http://hurault.perso.enseeiht.fr/asynchronousCommunication/`.

[3] Those are not the same as Dijkstra's history variables which are used to help/enable a correctness proof; here, history variables hold the necessary information on past events to realize deliveries with a specific ordering.

MODULE *peermanagement*

EXTENDS *Naturals*, *Sequences*
CONSTANT *BOTTOM_STATE*, *EMPTY_STATE*
VARIABLE *peers*

$PeersTypeInvariant \triangleq peers \in Seq(Nat)$
$trans(peer, init, next) \triangleq$ Peer transition from state *init* to next.
$\quad \wedge peers[peer] = init$
$\quad \wedge peers' = [peers \text{ EXCEPT } ![peer] = next]$

$AllPeersIn(states) \triangleq \forall i \in \text{DOMAIN } peers : peers[i] \in states$
$OnePeerIn(states) \triangleq \exists i \in \text{DOMAIN } peers : peers[i] \in states$
$NonBottom \triangleq \Box\neg OnePeerIn(\{BOTTOM\_STATE\})$
$Terminates \triangleq \Diamond\Box AllPeersIn(\{EMPTY\_STATE\})$

Peers state-based compatibility properties: no faulty reception and termination.

Fig. 9. TLA$^+$ Module for the Management of the Peers

MODULE *composition*

EXTENDS *Naturals*, *peermanagement*
CONSTANT *N*
VARIABLES *net*, *H*
$vars \triangleq \langle net, H, peers\rangle$
$Com \triangleq \text{INSTANCE } causal \text{ WITH } CHANNEL \leftarrow \{\text{“a”}, \text{“b”}\}$

$Init \triangleq \wedge Com!Init \wedge peers = \langle 11, 14\rangle$ $N = 2$, Initial states: **First peer:** $state_{11}$ **Second peer:** $state_{14}$

First peer: $a!.b!$
$t1(peer) \triangleq trans(peer, 11, 12) \wedge Com!send(peer, \text{“a”})$ **First peer:** $state_{11} \xrightarrow{a!} state_{12}$
$t2(peer) \triangleq trans(peer, 12, 13) \wedge Com!send(peer, \text{“b”})$ **First peer:** $state_{12} \xrightarrow{b!} state_{13}$

Second peer: $a?.b? + b?$
$t3(peer) \triangleq trans(peer, 14, 15) \wedge Com!receive(peer, \text{“a”}, \{\text{“b”}, \text{“a”}\})$ **Second peer:** $state_{14} \xrightarrow{a?} state_{15}$
$t4(peer) \triangleq trans(peer, 15, 16) \wedge Com!receive(peer, \text{“b”}, \{\text{“b”}\})$ **Second peer:** $state_{15} \xrightarrow{b?} state_{16}$
$t5(peer) \triangleq trans(peer, 14, 17) \wedge Com!receive(peer, \text{“b”}, \{\text{“b”}, \text{“a”}\})$ **Second peer:** $state_{14} \xrightarrow{b?} state_{17}$

$Fairness \triangleq \forall i \in 1 \mathinner{..} N : (\text{WF}_{vars}(t1(i)) \wedge \text{WF}_{vars}(t2(i)) \wedge \text{WF}_{vars}(t3(i)) \wedge \text{WF}_{vars}(t4(i)) \wedge \text{WF}_{vars}(t5(i)))$
$\quad \wedge \text{WF}_{vars}(Com!internal \wedge \text{UNCHANGED } peers)$
$Next \triangleq \exists i \in 1 \mathinner{..} N : (t1(i) \vee t2(i) \vee t3(i) \vee t4(i) \vee t5(i)) \vee (Com!internal \wedge \text{UNCHANGED } peers)$
$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge Fairness$

Fig. 10. Generated TLA$^+$ Module: $\xrightarrow{a!}\xrightarrow{b!}$ Composed with $\xrightarrow{a?}\xrightarrow{b?} + \xrightarrow{b?}$.

The communication models in Table 2 may involve redundant or unused information. For instance, a message in M^{sys}_{n-1} does not need to carry information about its sender: this model consists in ordering messages received on the same peer regardless of the emitting peer. In M^{sys}_{async} and M^{sys}_{rsc}, the role of the histories is limited to distinguishing messages and ensuring their uniqueness, and a simpler identifier would have sufficed. Similarly, in M^{sys}_{n-n}, a simple global counter is enough to implement it. The specification choices have been made to provide consistency between the communication models with the purpose of comparison.

One interesting point is that there are both “centralized” descriptions where a global shared history is used (M^{sys}_{n-n}, M^{sys}_{n-1}), and “distributed” descriptions, where each peer has its own history variable and no shared variable is necessary (M^{sys}_{rsc}, M^{sys}_{1-n}, M^{sys}_{causal}, M^{sys}_{1-1}, M^{sys}_{async}). This is conform to the observations in Section 2.4, where some models were classified as centralized. This is of importance for M^{sys}_{n-1}, which is sometimes displayed as a realistic asynchronous distributed model and is confused with M^{sys}_{1-1}.

model	vars	send $s_{cm} \xrightarrow{p,c!} s'_{cm}$	receive $s_{cm} \xrightarrow{p,c?,L} s'_{cm}$
M^{sys}_{rsc}	$net, \langle H_p \rangle$	$net = \emptyset$ $\wedge\ net' = \{\langle c, p, H_p\rangle\}$ $\wedge\ H'_p = H_p \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ \forall k : H'_k = H_k$
M^{sys}_{n-n}	net, H	$H' = H \cup \{\langle c, p, H\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H\rangle\}$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ \neg\exists \langle c_2, l, h_2\rangle \in net : \langle c_2, l, h_2\rangle \in h_1$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ H' = H$
M^{sys}_{1-n}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ \neg\exists \langle c_2, l, h_2\rangle \in net : l = j \wedge \langle c_2, l, h_2\rangle \in h_1$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ \forall k : H'_k = H_k$
M^{sys}_{n-1}	net, H	$H' = H \cup \{\langle c, p, H\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H\rangle\}$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ \neg\exists \langle c_2, l, h_2\rangle \in net : c_2 \in L \wedge \langle c_2, l, h_2\rangle \in h_1$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ H' = H$
M^{sys}_{causal}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ \neg\exists \langle c_2, l, h_2\rangle \in net : c_2 \in L \wedge \langle c_2, l, h_2\rangle \in h_1$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ H'_p = H_p \cup h_1 \cup \{\langle c_1, j, h_1\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$
M^{sys}_{1-1}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ \neg\exists \langle c_2, l, h_2\rangle \in net : l = j \wedge c_2 \in L \wedge \langle c_2, l, h_2\rangle \in h_1$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ \forall k : H'_k = H_k$
M^{sys}_{async}	$net, \langle H_p \rangle$	$H'_p = H_p \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ net' = net \cup \{\langle c, p, H_p\rangle\}$ $\wedge\ \forall k \neq p : H'_k = H_k$	$\exists \langle c_1, j, h_1\rangle \in net :$ $c_1 = c$ $\wedge\ net' = net \setminus \{\langle c_1, j, h_1\rangle\}$ $\wedge\ \forall k : H'_k = H_k$

(Initially, all vars are equal to $\emptyset$.)

Table 2. Formalization of the Communication Models. The send column contains the transition predicate for peer p to send a message on channel c; the receive column contains the transition predicate for peer p to receive a message on channel c while listening to the set of channels L.

Besides, it is also possible to count and/or limit the number of messages in transit, for the full network or on its projections (number of messages in transit for each channel). A counter of messages in transit is updated at send and receive transitions. This counter can be used to ascertain if the number of messages in transit is effectively bounded. When a bound is enforced at send transitions, it is used to quickly check if a system may be realizable. Note that this implies that the emission of a message is not always enabled.

4.4. Compatibility

In the following, we define what it means for a set of peers to be compatible. Compatibility properties and the required information they rely on are introduced.

4.4.1. States of Interest

In order to define compatibility criteria, we offer the possibility to mark subsets of states in each peer. We focus on two markings: terminal states and faulty states. Terminal states account for states that characterize a peer that has reached a point where the tasks it was supposed to perform are done. Faulty states designate states reached after an unexpected reception (that is to say a reception, imposed by the communication model, that is not correctly handled by a peer). Whether a state is terminal, faulty, or unmarked, is part of

MODULE *causal*

EXTENDS *Naturals*, *FiniteSets*
CONSTANTS *CHANNEL*, *N*
VARIABLES *net*, *H*
$Init \triangleq \land net = \{\} \land H = [i \in 1 \,..\, N \mapsto \{\}]$

$EmptyNetwork \triangleq net = \{\}$
$nochange \triangleq$ UNCHANGED $\langle net, H \rangle$
$internal \triangleq$ FALSE

$send(peer, chan) \triangleq$ Emission from peer of message #id on channel *chan*
LET $message \triangleq \langle chan, peer, H[peer] \rangle$ IN
$\land \; net' = net \cup \{message\}$
$\land \; H' = [H$ EXCEPT $![peer] = H[peer] \cup \{message\}]$

$receive(peer, chan, listened) \triangleq$ Reception on peer (while listening on channels in listened) of a message emitted on *chan*
$\exists \langle c1, p1, h1 \rangle \in net :$
$\land \; c1 = chan$
$\land \neg(\exists \langle c2, p2, h2 \rangle \in net : c2 \in listened \land \langle c2, p2, h2 \rangle \in h1)$
$\land \; net' = net \setminus \langle c1, p1, h1 \rangle$
$\land \; H' = [H$ EXCEPT $![peer] = H[peer] \cup h1 \cup \{\langle c1, p1, h1 \rangle\}]$

Fig. 11. TLA$^+$ Module Associated to the Causal Communication Model

the specification of the peer. The notion of terminal and faulty states implies that once a peer has reached such a state, it shall remain in it. No behaviour is specified after an unexpected reception (hence the term "unexpected") and it makes sense to affirm that a terminal state is indeed terminal. Then, it is not necessary to distinguished between different states with the same marking, and we collapse all the terminal states (resp faulty states) into one. We define the two universal peer states:

- 0 the terminal state,
- $\bot$ the faulty state.

4.4.2. Compatibility Properties

A compatibility property is given as an LTL formula over a system [MP92]. Let $System = (S, I, R)$ be a system with N peers. For a state $s = (s_1, \ldots, s_N, s_{CM}) \in S$, the following predicates are defined:

- $0_\forall \triangleq \forall p \in 1..N : s_p = 0$ (peers are all in the terminal state)
- $0_p \triangleq s_p = 0$ (termination of peer p)
- $\bot_\exists \triangleq \exists p \in 1..N : s_p = \bot$ (an unexpected message has been delivered)

The following compatibility properties are defined:

System termination The system always reaches the terminal state:
$System \models \Diamond\Box 0_\forall$

Peer termination The peer p always reaches the terminal state:
$System \models \Diamond\Box 0_p$

No faulty receptions No unexpected reception ever occurs:
$System \models \Box\neg\bot_\exists$

Absence of (Communication) Deadlock[4] : No state is stable except termination or fault:
$System \models \Diamond\Box 0_\forall \lor \Diamond\Box\bot_\exists \lor \forall s \in S : \Box\Diamond\neg s$

[4] Actually, we use TLC's native detection of deadlock, which checks ENABLED(*Next*). Note that *Next* may include user stuttering which is distinguished by TLC from the implicit stuttering of $[Next]_{vars}$.

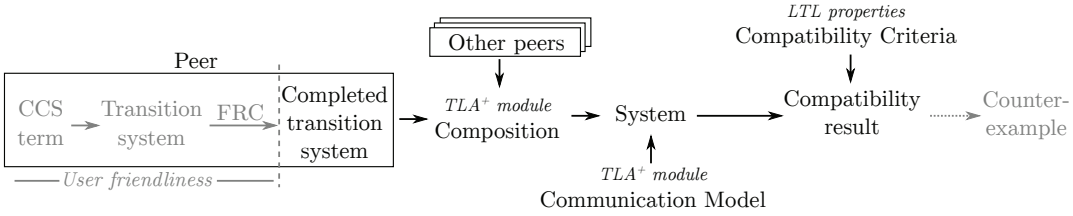


Fig. 12. Main Steps Performed by the Framework Including Optional User Friendly Steps

4.5. User Friendliness

Peers are specified with transition systems. Explicitly defining even quite simple peers can be cumbersome and one may want to step back and provide more abstract specifications. Furthermore, in order to check the compatibility of systems these peers take part in, information about terminal and faulty states has to be provided. For these reasons the proposed framework provides alternative ways to specify peers. In the following we will see how specifications of peers can be derived from CCS terms using the standard CCS rules and a transition completion step. These additional and optional steps are summed up in Figure 12.

4.5.1. Alternative Specification of Peer

A peer can alternatively be defined by a process specified with a CCS term where we consider:

- the empty process 0, neutral element of $+$ and $\|$,
- the prefixing operator $\cdot$, to perform an action followed by a process. An action is τ (an internal action), or $c!$ (a send action over a channel c), or $c?$ (a receive action on c),
- the choice operator $+$,
- the parallel composition operator $\|$,
- and process identifiers (defined by $X \triangleq \mathit{Process}$).

The peer transition system is derived from the CCS term using the standard CCS rules [Mil99, p.39] and excluding the synchronous communication rule REACT_t. Renaming and restriction are currently not used but they would have no impact on the verification of the compatibility. Since synchronous communication is not used, $\|$ is similar to an interleaving operator. It can model internal parallelism and dynamic creation of processes inside a given peer. The translation from a CCS term to a transition system is achieved through the Edinburgh Concurrency Workbench [CPS93]. We directly use the ability of The Concurrency Workbench to output the transition system of a CCS term (command `graph`, no change was required to the software). Each peer is independently translated, and as the translation is not applied to the composed system (the set of peers), synchronous communication (*reaction* in Milner's vocabulary) does not occur, and only *observable actions* (Milner's vocabulary) appear in the translation.

On-the-fly construction of the transition systems would make incompatibility detection more efficient as the complete transition system may be unnecessary for a counter-example, but proving compatibility would still require constructing all transitions of the peers. PlusCal specifications [Lam09], for instance, could also offer practical alternatives to CCS and the explicit generation of transitions.

4.5.2. Faulty Reception Completion

The faulty reception completion (FRC) consists in revealing the unexpected receptions in a peer and mark them as faulty. It adds the corresponding transitions and makes them point toward the faulty state (denoted $\perp$) introduced in 4.4.1. The completion ensures that the peer fits the intuitive viewpoint where the communication medium imposes messages.

The way transition systems are completed follows the definition of listened channels. Informally, for each state s where a receive transition exists, the future listened channels of s, i.e. the set of channels corresponding to possible future receptions, is computed. For each channel c in the future channels that is not already specified as an alternative choice in s, such a choice is provided by a transition towards $\perp$ and labeled by $c?$: $s \xrightarrow{c?} \perp$. These are called *faulty receptions*.

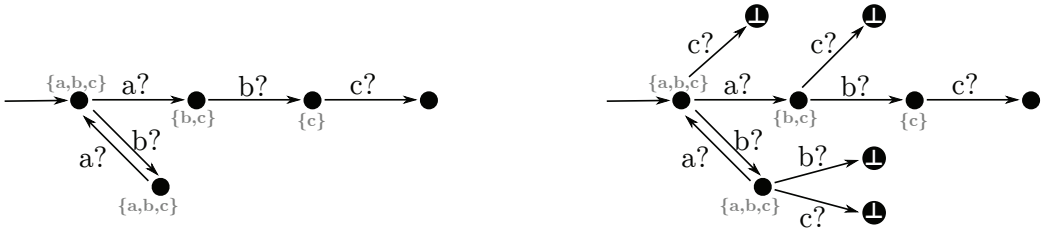


Fig. 13. Example of Faulty Reception Completion

Definition 4.5 (Faulty Reception Completion). Let $TS = (S, I, R, L)$ be a peer.
$FRC(TS) \triangleq (S \cup \{\bot\}, I, R \cup R_2, L)$ with $R_2 = \{s \xrightarrow{c?} \bot \mid s \in RS(S),\ c \in FC(s) \setminus LC(s)\}$, and

- $RS(S) \triangleq \{s \in S \mid \exists c \in C, s' \in S : s \xrightarrow{c?} s' \in R\}$
  (states having at least one receive transition)
- $FC(s) \triangleq \{c \in C \mid \exists s_1, s_2 \in S : s \to s_1 \in R^* \wedge s_1 \xrightarrow{c?} s_2 \in R\}$
  where R^* is the reflexive transitive closure of $\{(s_1, s_2) \in S^2 \mid \exists l \in L : s_1 \xrightarrow{l} s_2 \in R\}$.
  (future channels (possible receptions) in s)
- $LC(s) \triangleq \{c \in C \mid \exists s' \in S : s \xrightarrow{c?} s' \in R\}$ (see Definition 4.2)
  (listened channels in state s)

For instance, let us consider the peer represented on the left in Figure 13. The future channels are indicated next to each state. When there is no departing reception of a future channel, a faulty transition is added which results in the peer represented on the right. When composed with a peer $\xrightarrow{a!}\xrightarrow{b!}\xrightarrow{c!}$ and M_{1-1}^{sys}, the faulty receptions are impossible (because the send order is respected) and the peer always ends up in the far-right state (which may be of interest; e.g. 0 the terminal state). This cannot be guaranteed with M_{async}^{sys}, and the faulty state $\bot$ is reachable in the composition.

4.6. Composite Communication Models

Up to this point systems are composed of a set of peers associated to a communication model that ensures ordering properties on the communication medium. Messages transiting between the peers on all the channels involved in the communication are handled by a unique communication model. However, some practical cases require that different sets of channels be associated to different instances of communication models. This motivates the specification of composite communication models.

Messages on channels that are associated to only one model are emitted on (resp received from) one instance of that model. Messages on channels that are associated to several models are simultaneously emitted on (resp received from) instances of these models. Therefore, the reception of a message on such a channel can only occur when the ordering properties of all the involved communication models are met. For instance, if a, b, and c are channels associated to an instance of M_{causal}^{sys} and c, d to an instance of M_{1-1}^{sys}, then the reception of a message from c will require that it respects the causality of the emissions on a, b, and c, while also respecting the order of emissions on c and d from the same peer.

When a message is received from a channel, it has to be retrieved from every communication model instance it is associated to. In each of these instances, the same message can be locally identified by a different history. To prevent inconsistent receptions, messages are given a global id shared in the different instances.

Figure 14 shows the generated TLA$^+$ module associated to a composite communication model that takes into account channels that can be associated to the FIFO 1-1 communication model (channels exclusively in $CH1$), the causal communication model (channels exclusively in $CH2$), or both (channels in $CH1$ and $CH2$).

MODULE *multicom*

EXTENDS *Naturals*
CONSTANTS N, $CH1$, $CH2$
VARIABLES $messid$, $net1$, $H1$, $net2$, $H2$

$Var1 \triangleq \langle net1, H1 \rangle$ Two primitive communication models
$Var2 \triangleq \langle net2, H2 \rangle$
$Vars \triangleq \langle Var1, Var2 \rangle$

$Com1 \triangleq$ INSTANCE $fifo11$ WITH $CHANNEL \leftarrow CH1$, $net \leftarrow net1$, $H \leftarrow H1$
$Com2 \triangleq$ INSTANCE $causal$ WITH $CHANNEL \leftarrow CH2$, $net \leftarrow net2$, $H \leftarrow H2$

$ALLCHANNELS \triangleq CH1 \cup CH2$

$Init \triangleq messid = [chan \in ALLCHANNELS \mapsto \{\}] \land Com1!Init \land Com2!Init$

$max\_id(chan) \triangleq$ CHOOSE $n \in messid[chan] : \forall p \in messid[chan] : p \leq n$
$available\_id(chan) \triangleq$ IF $messid[chan] = \{\}$ THEN 1 ELSE (Message id generation and reuse
CHOOSE $n \in 1 \,..\, max\_id(chan) + 1 : n \notin messid[chan] \land (\forall p \in 1 \,..\, n : p = n \lor p \in messid[chan])$)

$EmptyNetwork \triangleq Com1!EmptyNetwork \land Com2!EmptyNetwork$
$nochange \triangleq$ UNCHANGED $messid \land Com1!nochange \land Com2!nochange$
$internal \triangleq \land$ UNCHANGED $messid$
$\land ((Com1!internal \land Com2!nochange) \lor (Com2!internal \land Com1!nochange))$

$send(peer, chan) \triangleq$
LET $id \triangleq available_id(chan)$IN
$\land messid' = [messid$ EXCEPT $![chan] = @ \cup \{id\}]$ Take available id
$\land$ On a channel from second group only: send on $Com2$
$\lor (chan \notin CH1 \land chan \in CH2 \land$ UNCHANGED $Var1 \land Com2!send(peer, chan, id))$
On a channel from first group only: send on $Com1$
$\lor (chan \in CH1 \land chan \notin CH2 \land$ UNCHANGED $Var2 \land Com1!send(peer, chan, id))$
On a channel from both group: send simultaneously (same message id) on $Com1$ and $Com2$
$\lor (chan \in CH1 \land chan \in CH2 \land Com1!send(peer, chan, id) \land Com2!send(peer, chan, id))$

$receive(peer, chan, listened) \triangleq$
$\exists id \in messid[chan] :$
$\land messid' = [messid$ EXCEPT $![chan] = @ \setminus \{id\}]$ Release message id for reuse
$\land$ On a channel from second group only: receive from $Com2$
$\lor (chan \notin CH1 \land chan \in CH2 \land$ UNCHANGED $Var1 \land Com2!receive(peer, chan, id, listened))$
On a channel from first group only: receive from $Com1$
$\lor (chan \in CH1 \land chan \notin CH2 \land$ UNCHANGED $Var2 \land Com1!receive(peer, chan, id, listened))$
On a channel from both group: receive simultaneously (same message id) from $Com1$ and $Com2$
$\lor (chan \in CH1 \land chan \in CH2$
$\land Com1!receive(peer, chan, id, listened) \land Com2!receive(peer, chan, id, listened))$

Fig. 14. Composite Communication Model with M_{1-1}^{sys} and M_{causal}^{sys}

The TLA$^+$ specifications of the primitive communication models are extended to carry the message identifiers.[5]

[5] These identifiers are not involved in the ordering properties of these models which are defined on the histories.

5. Validation of the Framework

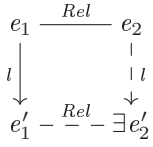
This section explains why the specifications of the models in the framework (M_*^{sys} of Section 4.3) are consistent with the descriptions of the concrete models (M_* of Section 2). We show that the hierarchy of the models is preserved in regard of refinement. We study both how M_*^{sys} are correct with regard to M_* (any generated execution with M_*^{sys} conforms to M_*), and how they are complete (M_*^{sys} generate all valid executions of M_*). The correctness is restricted to a class of systems (it requires that the peers have a certain regularity with regard to listened channels, as was previously hinted), and the completeness is restricted to a class of systems and executions.

5.1. Refinement Between Models

This section proves that the hierarchy of the communication models is preserved. This theorem is useful as the correctness and completeness results below are stated for systems/executions with additional constraints and are not universal, whereas this hierarchy of communication models is valid independently of the peers. We are comparing two labelled transitions systems, described with different variables, but using the same labels. The hierarchy is actually formed of refinements.

Definition 5.1 (Refinement of the Communication Models). Let M_1^{sys}, M_2^{sys} be two communication models. M_1^{sys} is a refinement of M_2^{sys} for the relation Rel iff (I, S, L, and R are the set of initial states, the set of states, the labels, and the transition relation as defined in Definition 4.3):

- $\forall e_1 \in I_{M_1^{sys}} : \exists e_2 \in I_{M_2^{sys}} : Rel(e_1, e_2)$: the initial states are in relation.
- $\forall e_1 \in S_{M_1^{sys}}, e_2 \in S_{M_2^{sys}}, e_1' \in S_{M_1^{sys}}, l \in L_{M_1^{sys}}$:
  $Rel(e_1, e_2) \wedge e_1 \xrightarrow{l} e_1' \in R_{M_1^{sys}} \Rightarrow \exists e_2' \in S_{M_2^{sys}} : Rel(e_1', e_2') \wedge e_2 \xrightarrow{l} e_2' \in R_{M_2^{sys}}$
  If two states are in relation, for any transition in the refined system, there must exist a transition in the original system.



Definition 5.2 ($\prec$). Let M_1^{sys}, M_2^{sys} be two communication models. We note $M_1^{sys} \prec M_2^{sys}$ if there exists Rel such that M_1^{sys} is a refinement of M_2^{sys} for the relation Rel.

Theorem 5.3 (Comparison of the Communication Models, with regard to Refinement).

- $M_{rsc}^{sys} \prec M_{n-n}^{sys} \prec M_{1-n}^{sys} \prec M_{causal}^{sys} \prec M_{1-1}^{sys} \prec M_{async}^{sys}$
- $M_{rsc}^{sys} \prec M_{n-n}^{sys} \prec M_{n-1}^{sys} \prec M_{causal}^{sys} \prec M_{1-1}^{sys} \prec M_{async}^{sys}$

Proof. The proofs of refinement are first order implications with quantifiers derived from Table 2. They are well adapted to SMT solvers, and they have been mechanized and checked with Why3 [Fil13] with the assistance of Alt-Ergo and CVC4. The proofs are available on http://hurault.perso.enseeiht.fr/asynchronousCommunication/refinements.why. A proof of refinement has also been conducted in the TLA$^+$ Proof System for similar models, requiring additional unique identifiers of the messages [CHMQ16]. □

5.2. Framework Executions vs Distributed Executions

Strictly speaking, distributed executions as considered in Section 2 are sequences of events; traces of the transition system built in Section 4.2 are sequences of states. We now need to relate these two kinds of executions[6].

Let us consider a maximal trace t of the transition system built in Definition 4.4, which thus satisfies $t_0 \in I \wedge \forall j \in \mathbb{N} : (t_j, t_{j+1}) \in R$ (I being the initial states and R being the transition relation; as R includes stuttering, the maximal traces are infinite). The corresponding execution is the sequence of transitions (t_j, t_{j+1}), for $j \in \mathbb{N}$. In the models M_*^{sys}, a transition is either an internal action, or a send transition, or a receive transition.

[6] To avoid ambiguity, we use the term *execution* for a sequence of events and for a sequence of transitions, and *trace* for a sequence of states. In transition systems, a maximal trace is commonly called an execution. Here, execution is the dual of trace.

Definition 5.4 (Framework Executions). For a set of peers S and a communication model M_*^{sys}, where $* \in \{async, 1{-}1, causal, 1{-}n, n{-}1, n{-}n, rsc\}$,
$Exec(S, M_*^{sys}) \triangleq \{dual(t) : t \text{ is a maximal trace of } S \text{ composed with } M_*^{sys}\}$
where $dual$ is the least fixed point solution to:
$dual(t) = event(t_0, t_1) \cdot dual(t^{+1})$, where t^{+1} is t with t_0 removed, and $\cdot$ is a sequence constructor

$$event(t_0, t_1) = \begin{cases} i & iff\ t_0 = t_1 \\ s(m)\ iff\ \exists p, \exists c, \exists m : send(t_0, t_1, p, c) \wedge \{m\} = net' \setminus net \\ r(m)\ iff\ \exists p, \exists c, \exists m : receive(t_0, t_1, p, c) \wedge \{m\} = net \setminus net' \end{cases}$$

For the communication models in Table 2, the function $event$ is deterministic and defined for all couples of consecutive states of a trace. This derives from the actions in Table 2 where the variable net is either unchanged (internal transition), or increased (send action), or decreased (receive action). Without ambiguity, the action is uplifted to get a label (internal action, send, receive). The corresponding execution σ of a maximal trace is a sequence of $\{i, s(m), r(m)\}$, where the messages are built by the framework according to M_*^{sys} rules.

Theorem 5.5 (Framework Executions are Distributed Executions). For a set of peers S and a communication model $M \in \{M_{async}^{sys}, M_{1-1}^{sys}, M_{causal}^{sys}, M_{1-n}^{sys}, M_{n-1}^{sys}, M_{n-n}^{sys}, M_{rsc}^{sys}\}, \forall \sigma \in Exec(S, M) : \sigma \in \mathcal{EXEC}$

Proof. The implementation of the communication models in the framework is based on history variables which are designed to ensure the uniqueness of messages (they always expand by appending new messages which they are part of), and therefore the uniqueness of communication events. In every model, before being received, a message needs to be added to the network. Since the network is empty in the initial state, a reception cannot occur before the associated emission. Thus the framework produces valid distributed executions. □

5.3. Correctness of the Framework

This section shows that the logical descriptions with histories realize the orders used to describe the concrete models, and lastly, that the execution generated with M_*^{sys} are valid with regard to M_*.

5.3.1. Histories Encode Orders

The next theorem links histories (as used by the framework) and orders (as defined in Section 2). Receive transitions in Table 2 include a predicate on the absence of another message which must be delivered first (the $\neg\exists\ldots$ line). With this theorem, these constraints match the delivery order predicates of Table 1.

Theorem 5.6 (Order Encoding). Histories correctly encode the orders $\prec_\sigma$, $\prec_{peer}$ or $\prec_{causal}$. For a set of peers S, an execution $\sigma \in Exec(S, M_*^{sys})$, and two distinct messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ sent in σ:

M_{n-n}^{sys} : $s(m_1) \prec_\sigma s(m_2) \Leftrightarrow m_1 \in h_2$
M_{n-1}^{sys} : $s(m_1) \prec_\sigma s(m_2) \Leftrightarrow m_1 \in h_2$
M_{1-1}^{sys} : $s(m_1) \prec_{peer} s(m_2) \Leftrightarrow m_1 \in h_2$
M_{1-n}^{sys} : $s(m_1) \prec_{peer} s(m_2) \Leftrightarrow m_1 \in h_2$
M_{causal}^{sys} : $s(m_1) \prec_{causal} s(m_2) \Leftrightarrow m_1 \in h_2$

where $s(m)$ is the send transition of m.

Proof.

- Cases M_{n-n}^{sys} and M_{n-1}^{sys}:
  - $\Rightarrow$: $s(m_1) \prec_\sigma s(m_2) \Rightarrow \exists m, n \in \mathbb{N} : s(m_1) = \sigma[m] \wedge s(m_2) = \sigma[n] \wedge m < n$. The global history H after sending m_1 (at time $m+1$) contains m_1. Since the history is increasing, H before sending m_2 (at time n) also contains m_1. h_2 is this history, so $m_1 \in h_2$.
  - $\Leftarrow$: $m_1 \in h_2$. Let n such that $s(m_2) = \sigma[n]$. Since the global history is initially empty, $\exists m \in \mathbb{N}, m < n$ such that m_1 is in the global history at time $m+1$ but m_1 is not in the global history at time m. So $s(m_1) = \sigma[m]$. Since $m < n$ then $s(m_1) \prec_\sigma s(m_2)$.

- Cases M^{sys}_{1-1} and M^{sys}_{1-n}:

  ⇒: $s(m_1) \prec_{peer} s(m_2) \Rightarrow (p_1 = p_2 \wedge \exists m, n \in \mathbb{N} : s(m_1) = \sigma[m] \wedge s(m_2) = \sigma[n] \wedge m < n)$. The peer history H_{p_1} after sending m_1 (at time $m+1$) contains m_1. Since histories are increasing, the peer history before sending m_2 (at time n) also contains m_1. h_2 is this history, so $m_1 \in h_2$.

  ⇐: $m_1 \in h_2$. Let n such that $s(m_2) = \sigma[n]$. $p_1 = p_2$ since in a peer history all messages are from the same peer. Since the peer histories are initially empty, $\exists m \in \mathbb{N}, m < n$ such that m_1 is in the peer history at time $m+1$ but m_1 is not in the peer history at time m. So $s(m_1) = \sigma[m]$. Since $m < n$ and $p_1 = p_2$ then $s(m_1) \prec_{peer} s(m_2)$.

- Case M^{sys}_{causal}: a history carries the causal past of a message, that is the set of messages which causally precede this message [BJ87, Bir96, KS11, Ray13]:

  – Send event: the current causal past of the peer is piggybacked in the message m, and the causal past of the next sent message from this peer will contain the message m;

  – Receive event: the causal past of the peer becomes the union of the current causal past of the peer, of the causal past of the received message (piggybacked in the message), and the message itself. Thus, the causal past of a future message from this peer will have all the messages which causally precede it, from the same peer or from peers which have, directly or indirectly, communicated with it.

  If $s(m_1) \prec_{causal} s(m_2)$, there is a causal path from $s(m_1)$ to $s(m_2)$, and m_2 contains m_1 in its causal past (its history). Conversely, if m_1 is in the history of m_2, it means m_1 is in the causal past of m_2, and thus that m_1 was sent causally before m_2.

□

5.3.2. *Stability with regard to Interest*

The stability with regard to interest indicates that, if a peer is not interested in a channel at a given time (i.e. it is not listening to it), then it will never be interested in it later: the set of listened channels can only decrease. This property is actually acquired when faulty receptions are added (faulty reception completion in Section 4.5.2) and is a natural property when asynchronous communication is present: violating it allows to arbitrarily anticipate or postpone a reception (especially a message which arrives at an unexpected point). Without stability, the peer could choose which messages it wants, and it is contrary to our assumption that it is the communication medium which pushes messages to the peers.

For instance, consider the peers $\{s_0 \xrightarrow{a!} s_1 \xrightarrow{b!} s_2,\ s_3 \xrightarrow{b?} s_4 \xrightarrow{a?} s_5\}$. This system is not stable w.r.t interest, as in state s_3, the peer is not listening to channel a, and later in state s_4, it starts to listen to a. The execution $\langle s(a), s(b), r(b), r(a)\rangle$ is allowed by the definition of M^{sys}_{1-1}, as the delivery on b is enabled in state s_3 (there is no other earlier message *on the listened channels*), but this execution does not conform to M_{1-1}. On the contrary, the completed peers are $\{\xrightarrow{a!}\xrightarrow{b!},\ \xrightarrow{b?}$ (with $\xrightarrow{a?} \bot$) $\xrightarrow{a?}\}$, this system is stable w.r.t. interest, and all its executions with any model M^{sys}_* conform to M_*.

Definition 5.7 (Stability w.r.t. interest). A peer (S, I, R, L) is stable w.r.t. interest iff $\forall s, s' \in S : s \to s' \in R^* \Rightarrow LC(s') \subseteq LC(s)$.

A system is stable w.r.t. interest iff all its peers are stable w.r.t. interest.

Lemma 5.8. For any peer P, the peer $FRC(P)$ is stable w.r.t. interest.

Proof. Let (S, I, R, L) be the transition system associated to $FRC(P)$. By the definition of FC, $\forall s, s' \in S : s \to s' \in R^* \Rightarrow FC(s') \subseteq FC(s)$, and by the construction of $FRC(P)$, $\forall s \in S : LC(s) = FC(s)$. □

5.3.3. *Correctness of the Models*

This section shows that each model of the framework (M^{sys}_*) conforms to its corresponding communication model (M_*). This often requires that the considered system is stable with regard to interest.

Theorem 5.9 (Correctness of the Models). For a set of peers S and a communication model M_*^{sys}, where $* \in \{async, 1{-}1, causal, 1{-}n, n{-}1, n{-}n, rsc\}$,

$Exec(S, M_*^{sys}) \subseteq \mathcal{E}xec(M_*)$.

For M_{n-1}^{sys}, M_{causal}^{sys} and M_{1-1}^{sys}, it is required that S is stable w.r.t. interest.

Proof.

- M_{rsc}^{sys} conforms to M_{rsc}: given the condition on the send and receive transitions of M_{rsc}^{sys}, a send transition can only occur if $net = \emptyset$, and a receive transition can only occurs if $net \neq \emptyset$. A send transition ensures that $net' \neq \emptyset$, and a receive transition ensures that $net' = \emptyset$. Thus, an execution in $Exec(S, M_{rsc}^{sys})$ alternates send and receive transitions (with interleaved τ) and conforms to M_{rsc}.
- M_{n-n}^{sys} conforms to M_{n-n}: let $\sigma \in Exec(S, M_{n-n}^{sys})$ and let us show that $\sigma \in \mathcal{E}xec(M_{n-n})$. Let two messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ such that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \wedge\ s(m_1) \prec_\sigma s(m_2)$, we need to prove (Definition 2.9) that $r(m_1) \prec_\sigma r(m_2)$.
  $s(m_1) \prec_\sigma s(m_2) \Longrightarrow_{Thm\ 5.6} m_1 \in h_2$
  By contradiction, assume that $r(m_2) \prec_\sigma r(m_1)$. When m_2 is received, m_1 is still in the network. Since $m_1 \in h_2$, Table 2 says that m_2 cannot be received while m_1 is in net. Contradiction. So $r(m_1) \prec_\sigma r(m_2)$.
- M_{n-1}^{sys} conforms to M_{n-1}: let $\sigma \in Exec(S, M_{n-1}^{sys})$ and let us show that $\sigma \in \mathcal{E}xec(M_{n-1})$. Let two messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ such that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \wedge s(m_1) \prec_\sigma s(m_2) \wedge peer(r(m_1)) = peer(r(m_2))$, we need to prove (Definition 2.10) that $r(m_1) \prec_\sigma r(m_2)$.
  $s(m_1) \prec_\sigma s(m_2) \Longrightarrow_{Thm\ 5.6} m_1 \in h_2$
  By contradiction, assume that $r(m_2) \prec_\sigma r(m_1)$. When m_2 is received, m_1 is still in the network. Since $m_1 \in h_2$, Table 2 ensures that c_1 is not of interest for the peer when m_2 is received. But later, when m_1 is received, c_1 is of interest for the same peer ($peer(r(m_1)) = peer(r(m_2))$). This is in contradiction with the stability of peers with regard to interest. So $r(m_1) \prec_\sigma r(m_2)$.
- M_{1-n}^{sys} conforms to M_{1-n}: let $\sigma \in Exec(S, M_{1-n}^{sys})$ and let us show that $\sigma \in \mathcal{E}xec(M_{1-n})$. Let two messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ such that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \wedge\ s(m_1) \prec_{peer} s(m_2)$, we need to prove (Definition 2.11) that $r(m_1) \prec_\sigma r(m_2)$.
  $s(m_1) \prec_{peer} s(m_2) \Longrightarrow_{Thm\ 5.6} m_1 \in h_2$
  $s(m_1) \prec_{peer} s(m_2) \Rightarrow p_1 = p_2$
  By contradiction, assume that $r(m_2) \prec_\sigma r(m_1)$. When m_2 is received, m_1 is still in the network. Since $m_1 \in h_2 \wedge p_1 = p_2$, Table 2 says that m_2 cannot be received while m_1 is in net. Contradiction. So $r(m_1) \prec_\sigma r(m_2)$.
- M_{causal}^{sys} conforms to M_{causal}: let $\sigma \in Exec(S, M_{causal}^{sys})$ and let us show that $\sigma \in \mathcal{E}xec(M_{causal})$. Let two messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ such that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \wedge\ s(m_1) \prec_{causal} s(m_2) \wedge peer(r(m_1)) = peer(r(m_2))$, we need to prove (Definition 2.12) that $r(m_1) \prec_\sigma r(m_2)$.
  $s(m_1) \prec_{causal} s(m_2) \Longrightarrow_{Thm\ 5.6} m_1 \in h_2$.
  By contradiction, assume that $r(m_2) \prec_\sigma r(m_1)$. When m_2 is received, m_1 is still in the network. Since $m_1 \in h_2$, Table 2 ensures that c_1 is not of interest for the peer when m_2 is received. But later, when m_1 is received, c_1 is of interest for the same peer ($peer(r(m_1)) = peer(r(m_2))$). This is in contradiction with the stability of peers with regard to interest. So $r(m_1) \prec_\sigma r(m_2)$.
- M_{1-1}^{sys} conforms to M_{1-1}: let $\sigma \in Exec(S, M_{1-1}^{sys})$ and let us show that $\sigma \in \mathcal{E}xec(M_{1-1})$. Let two messages $m_1 = \langle c_1, p_1, h_1\rangle$ and $m_2 = \langle c_2, p_2, h_2\rangle$ such that $r(m_1) \in \sigma \wedge r(m_2) \in \sigma \wedge\ s(m_1) \prec_{peer} s(m_2) \wedge peer(r(m_1)) = peer(r(m_2))$, we need to prove (Definition 2.13) that $r(m_1) \prec_\sigma r(m_2)$.
  $s(m_1) \prec_{peer} s(m_2) \Longrightarrow_{Thm\ 5.6} m_1 \in h_2$
  $s(m_1) \prec_{peer} s(m_2) \Rightarrow p_1 = p_2$
  By contradiction, assume that $r(m_2) \prec_\sigma r(m_1)$. When m_2 is received, m_1 is still in the network. Since $m_1 \in h_2 \wedge p_1 = p_2$, Table 2 ensures that c_1 is not of interest for the peer when m_2 is received. But later, when m_1 is received, c_1 is of interest for the same peer ($peer(r(m_1)) = peer(r(m_2))$). This is in contradiction with the stability of peers with regard to interest. So $r(m_1) \prec_\sigma r(m_2)$.
- M_{async}^{sys} conforms to M_{async}: M_{async} accepts any order.

□

5.4. Completeness of the Models

M_{async}^{sys} imposes no constraint on the delivery order of the messages and, for a set of peers S, it generates all possible asynchronous behaviors of S. The other models generate a subset of these executions. We show below that each model of the framework (M_*^{sys}) generates all the valid *fair* executions of its corresponding communication model (M_*).

Definition 5.10 (Fair Execution). A fair execution is an execution where all sent messages are eventually received:

$$\text{fair}(\sigma) \triangleq \forall m \in \mathcal{MES} : s(m) \in \sigma \Rightarrow r(m) \in \sigma$$

(when σ is a distributed execution, $r(m)$ necessarily occurs after $s(m)$)

Note: An execution which reaches the faulty state is considered fair: the $\perp$ state is stable and can empty the network by receiving and throwing away all the remaining messages in transit.

Let us consider the peers $S \triangleq \{\xrightarrow{a!}\xrightarrow{b!}\xrightarrow{c!}, \begin{array}{l}\xrightarrow{b?}\\ \searrow^{a?,c?}\end{array} \begin{array}{l}\xrightarrow{a?}\\ \searrow^{c?}\end{array}\}$. The execution $\langle s_1(a), s_1(b), r_2(b), s_1(c), r_2(c)\rangle$ is a valid execution with regard to M_{1-1} (it belongs to $\mathcal{E}xec(M_{1-1})$), but it does not belong to $Exec(S, M_{1-1}^{sys})$ because the message on b cannot be delivered while a message on a is available and of interest to the peer. The completeness with regard to fair executions is consistent with the intuition: an unfair execution is an execution where messages can be arbitrarily put on the side. In an unfair execution, any problematic message (i.e. an unexpected message whose delivery would invalidate the execution) can be ignored, and thus any send behavior could be turned into a valid receive behavior. Yet, the framework goal is to check if a set of peers are compatible, i.e. if their send behavior is compatible with their receive behavior.

Definition 5.11 (Single Receptor System). A Single Receptor System is a system composed of a set of peers $TS_p = (S_p, I_p, R_p, L_p), p \in 1..N$ where each channel is listened to by at most one peer ($LC(s)$ is the set of listened channels of state s, cf Definition 4.2, and is overloaded on peer):

$$\forall p, q \in 1..N, p \neq q : LC(TS_p) \cap LC(TS_q) = \emptyset \text{ where } LC(TS_p) \triangleq \bigcup_{s \in S_p} LC(s)$$

The completeness of some models requires this property. Consider the peers $\{\xrightarrow{a!}\xrightarrow{b!}, \begin{array}{l}\xrightarrow{a?}\\ \searrow^{b?}\end{array}, \xrightarrow{a?}\}$. The execution $\langle s_1(a), s_1(b), r_2(b), r_3(a)\rangle$ is a valid FIFO 1-1 (the messages on a and b are received on different peers), but this execution cannot be generated by the framework, as the reception on b is blocked by the message on a: there is no way to know that, *in the future*, the message on a will be received by another peer, and that the message on b can be delivered now. This problem occurs only because two peers are listening on a.

Theorem 5.12 (Completeness of the Models). For a set of peers S and a communication model M_*^{sys}, where $* \in \{async, 1-1, causal, 1-n, n-1, n-n, rsc\}$,

$$\forall \sigma \in Exec(S, M_{async}^{sys}) : \sigma \in \mathcal{E}xec(M_*) \wedge \text{fair}(\sigma) \Rightarrow \sigma \in Exec(S, M_*^{sys})$$

For M_{n-1}^{sys}, M_{causal}^{sys} and M_{1-1}^{sys}, it is required that S is a single receptor system.

Proof. M_{rsc}^{sys} is a special case and will be handled below. For the next paragraph, $*$ is any model except *rsc*.

The theorem is demonstrated by contradiction. Assume $\exists \sigma \in Exec(S, M_{async}^{sys}) : \sigma \in \mathcal{E}xec(M_*) \wedge \text{fair}(\sigma) \wedge \sigma \notin Exec(S, M_*^{sys})$. This means that σ could be generated with M_{async}^{sys} but not with M_*^{sys} while being an M_* execution. Let us show that such an execution does not exist.

By definition of $Exec(S, M_{async}^{sys})$, σ is the dual of a trace generated by S with M_{async}^{sys}. From the refinements (Theorem 5.3), a set of peers S composed with M_*^{sys} has no more transitions than S with M_{async}^{sys}. Thus, if $\sigma \notin Exec(S, M_*^{sys})$, there exists a state where a transition is enabled in S composed with M_{async}^{sys}, but, in the corresponding state of S composed with M_*^{sys}, the transition with the same label is disabled. Let's study this transition.

- τ transition: always enabled. Contradiction.

- Send transition: a send transition is always enabled (except for M_{rsc}^{sys} which is handled below). Thus it is enabled with M_*^{sys}. Contradiction.
- Receive transition: a receive transition on peer TS for a message $m_2 = (c_2, p_2, h_2)$ is disabled with M_*^{sys} if there exists at least one message which blocks the reception of m_2. There can only be finitely many blocking messages as they are in the past of m_2. Each such message $m_1 = (c_1, p_1, h_1)$ is in transit ($m_1 \in net$), is of interest to the peer ($c_1 \in LC(TS)$), and blocks the reception of m_2 ($m_1 \in h_2$ from the corresponding predicate of Table 2).
  - Case 1 : The message m_1 is never delivered in σ. Contradiction with σ is fair.
  - Case 2 : $r(m_1) \prec_\sigma r(m_2)$. After the delivery of m_1, $m_1 \notin net$. Contradiction.
  - Case 3 : $r(m_2) \prec_\sigma r(m_1)$.
    - Case M_{n-n}: $m_1 \in h_2 \Longrightarrow_{Thm\ 5.6} s(m_1) \prec_\sigma s(m_2) \Longrightarrow_{Def\ 2.9} r(m_1) \prec_\sigma r(m_2)$. Contradiction.
    - Case M_{n-1} (this requires that S is a single receptor system):
      $peer(r(m_1) = peer(r(m_2))$: $m_1 \in h_2 \Longrightarrow_{Thm\ 5.6} s(m_1) \prec_\sigma s(m_2) \Longrightarrow_{Def\ 2.10} r(m_1) \prec_\sigma r(m_2)$. Contradiction.
      $peer(r(m_1)) \neq peer(r(m_2))$: $c_2 \in LC(TS) \Longrightarrow_{Def.\ 5.11} c_1 \notin LC(TS)$. Contradiction.
    - Case M_{1-n}: $m_1 \in h_2 \Longrightarrow_{Thm\ 5.6} s(m_1) \prec_{peer} s(m_2) \Longrightarrow_{Def\ 2.11} r(m_1) \prec_\sigma r(m_2)$. Contradiction.
    - Case M_{causal} (this requires that S is a single receptor system):
      $peer(r(m_1) = peer(r(m_2))$: $m_1 \in h_2 \Longrightarrow_{Thm\ 5.6} s(m_1) \prec_{causal} s(m_2) \Longrightarrow_{Def\ 2.12} r(m_1) \prec_\sigma r(m_2)$. Contradiction.
      $peer(r(m_1)) \neq peer(r(m_2))$: $c_2 \in LC(TS) \Longrightarrow_{Def.\ 5.11} c_1 \notin LC(TS)$. Contradiction.
    - Case M_{1-1} (this requires that S is a single receptor system):
      $peer(r(m_1) = peer(r(m_2))$: $m_1 \in h_2 \Longrightarrow_{Thm\ 5.6} s(m_1) \prec_{peer} s(m_2) \Longrightarrow_{Def\ 2.13} r(m_1) \prec_\sigma r(m_2)$. Contradiction.
      $peer(r(m_1)) \neq peer(r(m_2))$: $c_2 \in LC(TS) \Longrightarrow_{Def.\ 5.11} c_1 \notin LC(TS)$. Contradiction.

Special case of M_{rsc}^{sys}: As $\sigma \in M_{rsc} \wedge fair(\sigma)$, σ alternates send and receive events (if σ weren't fair, messages could be sent and never received). net is initially empty, so in all states of the corresponding trace $|net| \leq 1$. If a receive transition is enabled with M_{async}^{sys}, it is enabled in the same state with M_{rsc}^{sys} (same action); as the next state ensures $|net| \leq 1$, if a send transition is enabled in a state with M_{async}^{sys}, the network is empty and the send transition with M_{rsc}^{sys} is enabled; a τ transition is always enabled with both models.
□

5.5. Conclusion on the Validation

This section has demonstrated the correctness and the completeness of the framework. For some models (M_{n-1}^{sys}, M_{causal}^{sys} and M_{1-1}^{sys}), this requires that the system is stable w.r.t. interest and is a single receptor system. The first point ensures that a peer cannot ignore a message now to postpone its reception at a later time (and thus invalidating the assumption that it is the communication model which pushes messages to the peer); the second one ensures that no knowledge of the future behavior of other peers is necessary to decide of a reception now. Completeness is only for fair or faulty executions as, again, an unfair execution could arbitrarily ignore messages.

6. Experiments and Results

6.1. Practical Example

Specification. Let us consider an examination management system composed of a student, a supervisor, a secretary, and a teacher. When the supervisor notices that a student has failed, he/she sends the name of the student to the teacher and the secretary, and the resit information to the student. If the student chooses to resit, he/she answers ok and asks the teacher for the exam. The teacher sends the needed materials and

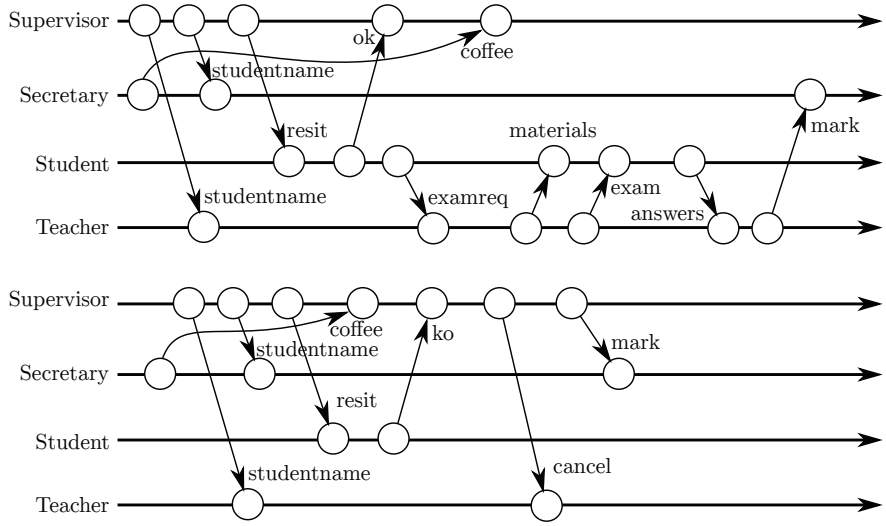


Fig. 15. Examples of Expected Executions

$$
\begin{aligned}
\textbf{Supervisor} &\triangleq \mathit{studentname}! \cdot \mathit{studentname}! \cdot \mathit{resit}! \cdot ((\mathit{ok}? \cdot 0 + \mathit{ko}? \cdot \mathit{cancel}! \cdot \mathit{mark}! \cdot 0) \| (\mathit{coffee}? \cdot 0)) \\
\textbf{Secretary} &\triangleq \mathit{coffee}! \cdot \mathit{studentname}? \cdot \mathit{mark}? \cdot 0 \\
\textbf{Student} &\triangleq \mathit{resit}? \cdot (\tau \cdot \mathit{ko}! \cdot 0 + \tau \cdot \mathit{StudentOK}) \\
\mathit{StudentOK} &\triangleq \mathit{ok}! \cdot \mathit{examreq}! \cdot \mathit{materials}? \cdot \mathit{exam}? \cdot \mathit{answers}! \cdot 0 \\
\textbf{Teacher} &\triangleq \mathit{studentname}? \cdot (\mathit{cancel}? \cdot 0 + \mathit{examreq}? \cdot \mathit{TeacherExam}) \\
\mathit{TeacherExam} &\triangleq \mathit{materials}! \cdot \mathit{exam}! \cdot \mathit{answers}? \cdot \mathit{mark}! \cdot 0
\end{aligned}
$$

Fig. 16. Supervisor-Secretary-Student-Teacher Specification

then the exam, after which the student sends back his/her answers, then the teacher sends a mark to the secretary. If the student declines to resit, he/she informs the supervisor who sends a cancel message to the teacher and the former mark to the secretary. An unrelated exchange also occurs between the secretary and the supervisor who would like to meet during the coffee break. The secretary sends a message to inform the supervisor that coffee is ready. The supervisor is ready to join after he/she has sent work-related messages: just before, after, or during the time he deals with the student's choice. Sample executions are depicted in Figure 15 and the system is specified in Figure 16.

Next, consider the properties needed to make this work as intended. There is a causal dependency between the *studentname* message and the *examreq* message (the request for the exam must not arrive before the student name). This causal dependency comes from the *resit* message, which follows the *studentname* message and is the cause of the *examreq* message. Causal communication is thus required. Moreover, if a *cancel* message is sent, it should be received after the student's name by the teacher. Therefore, *cancel* is part of this causal group. The same holds for the *mark* channel, since the secretary first expects a *studentname*. Besides, the *materials* and the *exam* are sent in two separate messages and are expected to be received in this order by the student. Lastly, the coffee break exchange requires that several messages can be in transit so that the supervisor can send the *studentname* and *resit* messages after the secretary has sent *coffee*.

This example is checked with the seven asynchronous models of Table 2, and with the following composite model $M^{sys}_{composite}$:

$$
\begin{aligned}
M^{sys}_{causal} &: \{\mathit{studentname}, \mathit{resit}, \mathit{examreq}, \mathit{cancel}, \mathit{mark}\} \\
M^{sys}_{1-1} &: \{\mathit{materials}, \mathit{exam}\} \\
M^{sys}_{async} &: \{\mathit{ok}, \mathit{ko}, \mathit{answers}, \mathit{coffee}\} \quad \text{(no constraint)}
\end{aligned}
$$

Compatibility. In this example, *studentname* is a channel over which two messages are sent and from which they are received by different services (teacher and secretary). In addition, *mark* is a channel over which only one message is to transit, but it may be emitted by different services (supervisor and teacher). Therefore,

	M^{sys}_{rsc}	M^{sys}_{n-n}	M^{sys}_{1-n}	M^{sys}_{n-1}	M^{sys}_{causal}	M^{sys}_{1-1}	M^{sys}_{async}	$M^{sys}_{composite}$
Termination	×	✔	✔	✔	✔	×	×	✔
Termination with an empty network	×	✔	✔	✔	✔	×	×	✔
Partial termination (secretary)	×	✔	✔	✔	✔	×	×	✔
No faulty receptions	✔	✔	✔	✔	✔	×	×	✔
Absence of communication deadlock	×	✔	✔	✔	✔	✔	✔	✔

Fig. 17. Compatibility Results

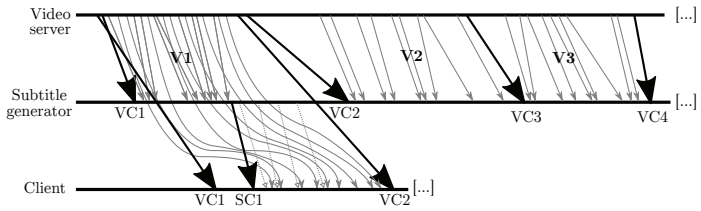


Fig. 18. Transmission of One Part Between the Three Peers

compatibility, especially termination of the secretary service, is not trivial. Consequently, in addition to the generic compatibility properties defined in 4.4, we also consider the termination of the secretary and we check if all messages have been received upon full termination.

Figure 17 presents the results. It confirms that causality (or a model stricter than causality) is needed to ensure compatibility of the composition. However, causality is not required over the whole set of channels. The composite model with the considered partition is a restrictive enough communication model. In this example, the maximal number of distinct states is 988.

6.2. Composite Communication Model: Use Case

The previous example has illustrated how channels can be partitioned and associated to different communication models in order to perform sharper analysis. Here we provide a case where the need for channels to be associated to more than one communication model arises.

Let us consider a system where a client watches a live video with subtitles. The video is stored on a remote video server, and captioning is performed on the fly by a subtitle generator. The streams are cut into video parts (resp. subtitle parts) denoted Vi (resp. Si) where i designates the i-th part. The client expects to receive each video part, one after the other, and the associated subtitle part with little enough delay between them. In order to achieve that goal, we introduce checkpoints in the streams. Each video or subtitle part Vi (resp. Si) is preceded by the emission of an associated checkpoint message denoted VCi (resp. SCi). FIFO 1-1 ordering is used so that messages that compose a part are received in the right order, as well as the checkpoint messages. Figure 18 provides a possible execution where a part is transmitted as expected.

FIFO 1-1 communication is not sufficient to prevent the video stream from being late with regard to the subtitles, as they are sent by different peers (Figure 19). Causal communication on the entire system would

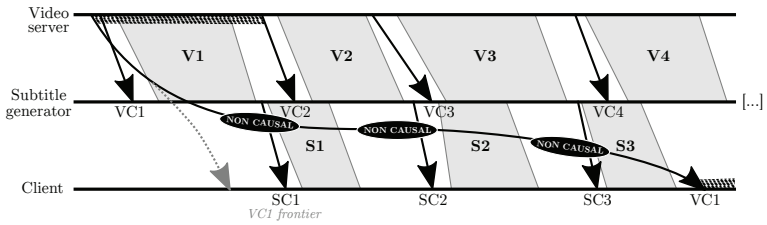


Fig. 19. Desirable and Undesirable Executions with FIFO 1-1

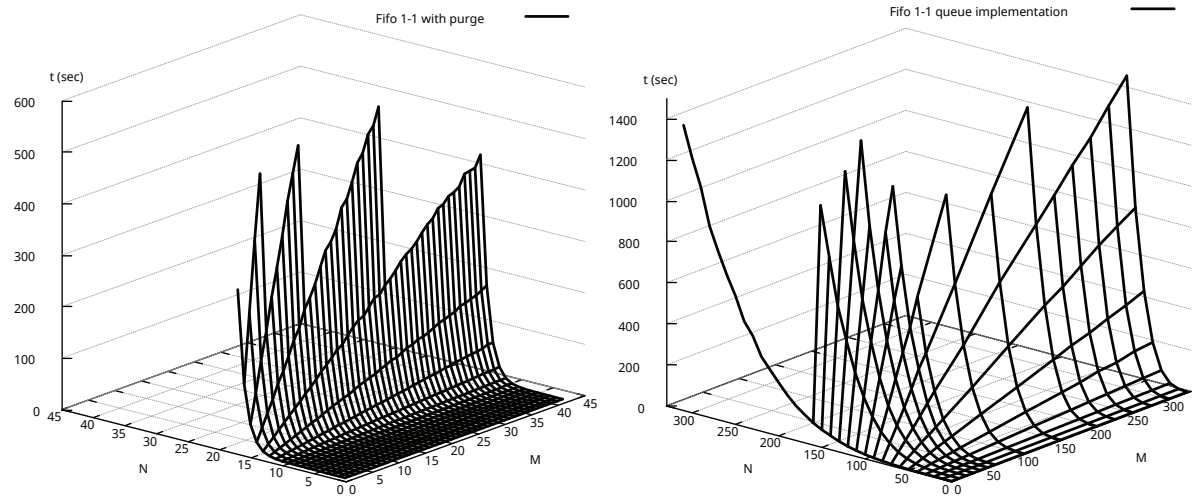


Fig. 20. Optimized Implementations

prevent this, but this is excessive, as a little desynchronization between a video part and its corresponding subtitle part is not perceived. It is sufficient to consider a composite communication model, where all the channels are associated to an M_{1-1}^{sys} instance, and all the control point channels are also associated to an M_{causal}^{sys} instance. Model checking of this system (three parts of two V_i/S_i messages each, plus checkpoint messages, for a total of 30 messages) generates 353988 distinct states and takes around $22s$ with an optimized implementation of M_{1-1}^{sys}.

6.3. Benchmarking

6.3.1. Optimized Implementations of the Communication Models

Several practical issues and restrictions arise from the logical implementations of the communication models M_*^{sys}. For instance, as the presented communication models are based on message histories that never decrease, we are confronted to performance limitations. Moreover, systems involving behaviors with loops, such as for instance the trivial system composed of any communication model and two peers derived from the CCS terms $P_1 \triangleq a!.b?.P_1$ and $P_2 \triangleq a?.b!.P_2$, will actually consist in an infinite transition system.

A mechanism to purge history has been implemented, where messages that are retrieved from the network of a communication model are recursively removed from histories. This purge reduces the previous example to a finite state system over which model checking always reaches a conclusion. These alternative communication model specifications (denoted $M_{*,purge}^{sys}$) have proven useful to check practical examples. When possible, more optimized and practical implementations (denoted $M_{*,impl}^{sys}$) consist in using message counters instead of message histories (for instance, in M_{1-1}^{sys}), or explicit sequences (for instance, for M_{n-n}^{sys}). However, these implementations are not guaranteed to be equivalent to the logical specifications.

6.3.2. Scenario

We consider two parameters $n \geq 0$ and $m \geq 0$, $a_1, \ldots, a_n$ and b channels. Two peers are specified by the following CCS terms from which we derive transition systems and apply FRC:

$(a_1!.\ldots.a_n!.b?)^m$ and $(a_1?.\ldots.a_n?.b!)^m$

The composition consists in transmitting sequences of n messages (on the a_i channels) from the first to the second peer. A synchronization message is exchanged (from the second to the first peer) between each sequence. The messages are expected to be received in the order of their emission and a communication model has to ensure that unexpected receptions are impossible.

The framework is used to check the termination of the proposed composition. In the following results,

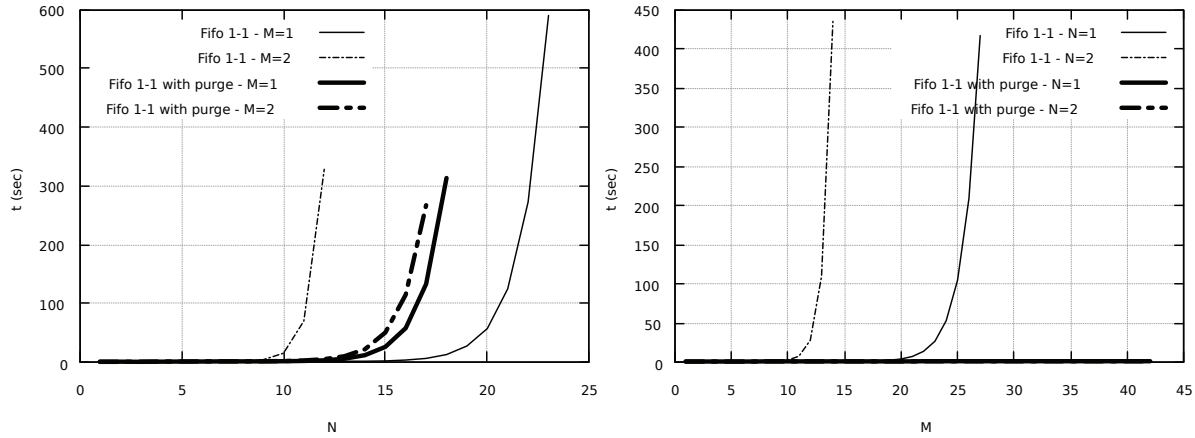


Fig. 21. Comparison Between the Formal Implementation and the Purge-Based Optimized Implementation

m	1	1	41	51	101	301
n	1	311	111	51	91	1
states	9	48834	259494	70334	432184	1209
transitions	11	97041	509801	135361	845781	1211

Table 3. Number of States and Transitions Using M_{1-1}^{sys} (Samples)

we consider M_{1-1}^{sys}, an alternative specification with history purge $M_{1-1,purge}^{sys}$, and an implementation using a queue $M_{1-1,impl}^{sys}$. The machine that runs the simulations is 2×4 cores Intel Xeon CPU E5-2690 v3 at $2.60GHz$ with $23GiB$ of memory. We focus on the number of generated states and the time required to perform model checking. The results are presented in Figures 20 and 21 and Table 3 for M_{1-1}^{sys}, $M_{1-1,purge}^{sys}$, and $M_{1-1,impl}^{sys}$. The results obtained with the other models of Table 2 and their different implementations are similar. Indeed, their specification do not differ much in terms of data structure and guard on reception, and the time required to explore the state space do not vary significantly. Differences appear between the different implementations of a given model.

6.3.3. Analysis

With $M_{1-1,purge}^{sys}$, the results show that the number of states and runtime linearly increase with m the number of critical sequences. They exponentially increase when it comes to n. It accounts for the maximum number of messages in transit at a given time and all the possible receptions that have to be visited. m corresponds to the number of repetitions of the scenario, thus the linear profile. Runtime also grows exponentially for m with M_{1-1}^{sys} because histories of past iterations accumulate. Checking if a reception is possible requires to explore this entire past whereas $M_{1-1,purge}^{sys}$ and $M_{1-1,impl}^{sys}$ do not keep track of received messages.

	Advantages	Drawbacks
M_{*}^{sys}	Direct translation of the formal specification.	Keeps track of every emitted message: poor m-type scalability.
$M_{*,purge}^{sys}$	Close to the formal specification. No accumulation of messages in histories. Better m-type scalability (linear).	Additional time needed to purge histories: state space explosion occurs earlier in terms of maximum number of messages in transit.
$M_{*,impl}^{sys}$	Way better overall performances. Useful as a quick first compatibility check or to find potential counterexamples.	Equivalence to the formal specification is not guaranteed.

7. Related Work

7.1. Distributed System

Ordered delivery has long been studied in distributed algorithms and goes back to Lamport's paper introducing logical clocks [Lam78]. Implementations of the basic communication models (M_{causal}, M_{1-1}) using histories or clocks are explained in classic textbooks [BM93, CDK94, Tel00, KS11, Ray13], and the minimum required information to realize these orders has been studied [PBS89, KS98]. Moreover, asynchronous communication models in distributed systems have been studied and compared in [KS11] (notion of ordering paradigm) and [CBMT96] (notion of distributed computation classes) using a similar approach based on distributed executions and the causal order on communication events [Lam78]. They both show the hierarchy of classic communication models such as non-FIFO, FIFO or causally-ordered that respectively correspond to M_{async}, M_{1-1} and M_{causal}.

Our work is complementary in two ways. First, we consider additional distributed communication models, namely M_{n-n}, M_{1-n} and M_{n-1}, which are of interest since they are not totally ordered. M_{n-1} for instance, the FIFO order with instantaneous delivery, is sometimes used in the literature [OSB13] without distinction from the classic FIFO order. Then, we consider another approach for comparison that involves system specifications using transition systems to show how, in terms of compatibility checking, classic execution-based ordering between the models [CBMT96, KS11] is preserved and how the additional commonly used models fit into that ordering by matching the causal model.

Tel's textbook [Tel00] describes a distributed system as a *collection of processes and a communication subsystem*. Each process is a transition system, and the transition system induced under asynchronous communication is built with the product of the process transition systems extended with a collection of messages in transit, and two rules for send and receive. His formal definition considers synchronous and fully asynchronous (unordered) communication. FIFO (M_{1-1}) and causal communication are mentioned but are not formalized. Tel's goal is to describe distributed algorithms, whereas our objectives is to study communication models. Our work departs from his by explicitly describing the communication subsystem with a transition system, by considering several orderings, by comparing these communication models, and by offering a framework for the verification of asynchronously communicating peers based on these models.

7.2. Compatibility Checking

Compatibility of services / software components has largely been studied, with two main goals: Can services communicate and provide more complex services? And can one service be replaced by another one (substitutability)? These two notions of compatibility are different. In the first case, the services must be complementary, whereas in the second case they should provide the same functionality. Classically, either the notion of simulation (as in [ABDF08]) or the notion of trace inclusion (as in [CLB08]) is used to express this sameness. In this taxonomy, we can also include different models of failure traces [GGH+10], where refusal sets may be used to model (preservation of) process receiving capabilities and therefore absence of forever pending messages. We are mainly interested in the first problem. Many approaches exist to verify behavioral compatibility of web services or software components.

Different formalisms are used to represent the services: finite-state machines [DOS12, CLB08, BCT04, FUMK04], process algebra [DWZ+06, BCPV04, CPT01], Petri nets [LFS+11, TFZ09, Mar03]. Different criteria are used to represent compatibility: deadlock freedom [DOS12, FUMK04], unspecified receptions [BZ83, DOS12], at least one execution leads to a terminal state [DOS12, BCT04, DWZ+06, LFS+11], all the executions lead to a terminal state [BCT04, BCPV04], no starvation [FUMK04], divergence [BCPV04]. Domain application conditions are also used [CLB08, CPT01]. The communication models used are synchronous [DOS12, BCT04, FUMK04, DWZ+06, BCPV04, CPT01] or FIFO n-1 [BBO12, OSB13].

On the specific point of stability w.r.t. interest (Definition 5.7) and faulty reception completion (Section 4.5.2), this is reminiscent of Brand and Zafiropulo's unspecified reception approach [BZ83]. In their work, if a state can receive a given message, then a successor state (accessible via send events) must also accept this message. In other words, for a system to be correct w.r.t. unspecified reception (and thus for compatibility), if a message can be received at a given state, its reception must also be specified at later states. In our work, we reverse the proposition: if a message can be received at a given state, the communication model may deliver it earlier and the system must expect this situation. This is the stability w.r.t.

interest property. The faulty reception completion ensures that fault transitions are introduced to get this stability property.

To sum up, although some works are dedicated to several compatibility criteria, all of them are dedicated to one communication model, mostly the synchronous model. None of them proposes a verification parameterized by both the compatibility criteria and multiple communication models. Moreover, only a few approaches also provide a tool to automatically check the proposed composition. Compared to these works, we propose a unified formalization of several communication models and compatibility criteria, and a framework which allows to check the correctness of a composition in a unified manner, using any combination of the communication models. Lastly, the prototype tool returns an invalid execution counterexample when a compatibility criterion is not met.

7.3. System Description

7.3.1. I/O Automata

Input/output automata [Lyn96] provide a generic way to describe components that interact with each other thanks to input and output actions. Those actions are partitioned into tasks over which fairness properties can be defined in the same way fairness properties can be set over TLA$^+$ actions. Components can either describe processes or communication channels. They can also be composed and some output actions can be made internal (hiding) in order to specify complex systems. I/O automata are said to be *input-enabled*: every input action of an automaton is required to be enabled in every state, in order to avoid "the failure to specify what the component does in the face of unexpected inputs" [Lyn96, p. 203]. In our work, the faulty reception completion (Section 4.5.2) and the requirement of stability with regard to interest (Definition 5.7) play a similar role: the absence of unexpected messages is equivalent to the unreachability of the faulty state. I/O automata can model asynchronous systems in a broad sense and provide a powerful framework to describe distributed systems. However, few automatic tools have been developed to make use of IO automata and perform modeling and property checking.

7.3.2. Process Calculi

One of the interest of process calculi is their algebraic representation which is simple, concise and powerful. The processes are described by a term under an algebra. They are constructed from other processes thanks to composition operators (parallel composition, sequence, alternative, ...). The basic processes represent elementary actions, which are most often communication operations (send or receive).

CCS [Mil82] is an early and seminal calculus that we have chosen for its simplicity and user friendliness to describe peers. Its main disadvantage for our work is that communication is synchronous. Milner has also defined the π-calculus [Mil99]. The main difference is the introduction of parameters: channels can be communicated through channels themselves. This allows to describe systems with dynamic configurations. Still, the π-calculus is also synchronous. Nevertheless, some adaptation of the π-calculus have been proposed [HT91, Bou92] to change the semantics of communication into an asynchronous one. The asynchronous π-calculus represents asynchronous communication by using a send primitive with no continuation. In [Pal03], Palamidessi proves that this asynchronous π-calculus is less expressive than the full π-calculus. [BPV08] compares the asynchronous π-calculus with three different versions of the π-calculus where channels are explicitly represented as special buffer processes. Those buffers are bags, queues or stacks depending on the model of interest. There is a strong correspondence between the asynchronous model and the model with bags. But that correspondence does not hold with queues and stacks. This is conform to our result of non equivalence of our pure asynchronous model with any of our FIFO models. Since we are interested in comparing the communication models used by the distributed computing community, their π-calculus with queue is not satisfactory. Indeed, the FIFO property is guaranteed for a given channel, whereas distributed algorithms use this property between peers. Since two peers can communicate together via several channels, there is no direct correspondence between these two approaches. We haven't studied communication models using stacks, since those models are not relevant when dealing with distributed applications, but it would be easy to give a TLA$^+$ specification as the dual of FIFO models, using stacks instead of queues.

Richer process calculi exist, such as the Join-calculus [FG96] (and its extension to mobility [FGL$^+$96]) based on the reflexive CHAM (CHemical Abstract Machine) [BB92] and also the Ambient calculus [CG98].

They allow the description of separated membranes/domains, where processes interact with each other within a domain or perform explicit actions to move in or out of domains. These calculi are mainly used to model mobility, distribution, firewalls and security properties. But they are not fitted to our concerns for two reasons. Firstly, modelling distribution is not straightforward (usually a mix of local communications and moves between domains) whereas we want to keep it as simple as possible, as distribution is at the core of our concerns. Secondly, they are not parameterized over communication models and directly encoding them would also be cumbersome.

There exist several model checkers for process calculi. CADP [GLMS13] analyzes high-level descriptions written in various languages with synchronous communication, such as LOTOS, LNT (LOTOS New Technology) [SCG+00], FSP... We can also cite SPIN [Hol04] for Promela [Hol04, chap. 3], where communication are realized using FIFO message channels. Our goal is not to model check process calculus. Our main concern is the comparison of communication models and being able to check properties over a system with several communication models (including composite ones).

8. Conclusion and Perspectives

This paper considers the diversity of point to point asynchronous communication models, more complex but more realistic than synchronous communication, and provides two points of view to their analysis. Seven models are exposed, some classic and some less common. In particular, the notion of FIFO communication is clarified and the variants it covers are presented. All these communication models are specified as properties on distributed executions involving orders on communication events (sending and receiving messages), including Lamport's causal order. A hierarchy between the models arises from the inclusion of these orders.

Then, a framework to check compatibility of asynchronous communicating peers is presented. The composition can use different communication models (e.g. FIFO or causal) for different groups of channels. The description of the communication models is operational but is abstract enough to not preclude realistic implementations. The conformance of the framework with the execution-based specifications of the communication models is shown: correctness and completeness are proven.

The peers and the communication models are defined with transition systems and the composition is close to a synchronous product of the peers and the model. Our framework is also parametric with regard to the compatibility criteria, specified as LTL properties. The framework has been instantiated in TLA$^+$ and thus benefits from its tools, especially the TLC model checker. Additionally, TLA$^+$ specifications of the peers can be automatically generated from a high-level behavioral description.

On-going work aims at extending the asynchronous models, introducing broadcast (analogous to a message consumed by more than one peer) and communication failures (message loss). A second point of interest is to find the weakest communication model (in the sense of less restrictive according to the established hierarchies) required to achieve compatibility. Currently, the designer specifies which communication model is used for each channel. Then, compatibility can be verified. It would be interesting to automatically discover the right partitioning and the weakest communication models for these partitions.

In terms of performance, the experiments show that practical implementations of communication models push the boundaries of model checking beyond what can be expected from direct implementations of the specifications. The conformity of such practical implementations with the formal specifications is not guaranteed yet. Extending the variety of implementations (close to specification, realistic, or performance-oriented) and proving their correctness and completeness would allow for more efficient and more reliable compatibility checking.

Lastly, we have observed that, for some stable global properties of a system, e.g. occurrence of unexpected messages, some communication models always yield the same compatibility answer: the sets of executions for these models are different but the property is inevitable in all of them. A precise characterization of this class of the properties would allow to switch from a communication model to a weaker one without having to prove again the compatibility.

Acknowledgements We would like to thank the anonymous reviewers for their careful reading of the paper and their invaluable comments. We are grateful to Xavier Thirioux for various discussions and for his shared expertise in formal methods.

References

[ABDF08] Ali Ait-Bachir, Marlon Dumas, and Marie-Christine Fauvet. BESERIAL: Behavioural Service Analyser. In *Business Process Management International Conference. Demo session.*, pages 374–377, 2008. LNCS 5240.

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

[BBO12] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Synchronizability for verification of asynchronously communicating systems. In *13th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'12, pages 56–71. Springer-Verlag, 2012.

[BCPV04] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electronic Notes in Theoretical Computer Science*, 105:73–94, December 2004.

[BCT04] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling – ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2004.

[Bir96] Kenneth P. Birman. *Building secure and reliable network applications*. Manning, 1996.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, January 1987.

[BM93] Özalp Babaoğlu and Keith Marzullo. Consistent global states of distributed systems : Fundamental concepts and mechanisms. In Sape J. Mullender, editor, *Distributed Systems*, pages 55–96. ACM Press Frontier Series, second edition, 1993.

[Bou92] Gérard Boudol. Asynchrony and the Pi-calculus. Research Report RR-1702, INRIA, 1992.

[BPV08] Romain Beauxis, Catuscia Palamidessi, and Frank D. Valencia. On the asynchronous nature of the asynchronous π-calculus. In Rocco De Nicola, Pierpaolo Degano, and José Meseguer, editors, *Concurrency, Graphs and Models*, Lecture Notes in Computer Science, pages 473–492. Springer, 2008.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, April 1983.

[CBMT96] Bernadette Charron-Bost, Friedemann Mattern, and Gerard Tel. Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4):173–191, February 1996.

[CDK94] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: concepts and design*. Addison Wesley, second edition, 1994.

[CF99] Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Transaction on Parallel and Distributed Systems*, 10(6):642–657, June 1999.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *First International Conference on Foundations of Software Science and Computation Structure*, FoSSaCS '98, pages 140–155. Springer-Verlag, 1998.

[Cha97] Michel Charpentier. A UNITY mapping operator for distributed programs. In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *4th Int'l Symposium of Formal Methods Europe (FME'97)*, volume 1313 of *Lecture Notes in Computer Science*, pages 665–684. Springer-Verlag, September 1997.

[CHMQ16] Florent Chevrou, Aurélie Hurault, Philippe Mauran, and Philippe Quéinnec. Mechanized refinement of communication models with TLA+. In *International ABZ Conference*, page 6. Springer-Verlag, May 2016.

[CHQ15] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. Automated verification of asynchronous communicating systems with TLA+. *Electronic Communications of the EASST (PostProceedings of the 15th International Workshop on Automated Verification of Critical Systems)*, 72, 2015.

[CL85] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[CLB08] Heung Seok Chae, Joon-Sang Lee, and Jung Ho Bae. An approach to checking behavioral compatibility between web services. *International Journal of Software Engineering and Knowledge Engineering*, 18(2):223–241, 2008.

[CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[CPT01] Carlos Canal, Ernesto Pimentel, and José M. Troya. Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41(2):105–138, October 2001.

[DOS12] Francisco Durán, Meriem Ouederni, and Gwen Salaün. A generic framework for n-protocol compatibility checking. *Science of Computer Programming*, 77(7-8):870–886, July 2012.

[DSU04] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36:372–421, December 2004.

[DWZ+06] Shuiguang Deng, Zhaohui Wu, Mengchu Zhou, Ying Li, and Jian Wu. Modeling service compatibility with pi-calculus for choreography. In *25th International Conference on Conceptual Modeling*, Conceptual Modeling - ER 2006, pages 26–39. Springer-Verlag, 2006.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*, POPL '96, pages 372–385. ACM, 1996.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, 1996.

[Fil13] Jean-Christophe Filliâtre. One logic to use them all. In *24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Artificial Intelligence*, pages 1–20, Lake Placid, USA, June 2013. Springer.

[FUMK04] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Compatibility verification for web service choreography. In *IEEE International Conference on Web Services*, pages 738–, 2004.

[GGH+10] Paul Gardiner, Michael Goldsmith, Jason Hulance, David Jackson, Bill Roscoe, Bryan Scattergood, and Philip Armstrong. FDR2 user manual. Technical report, Oxford University, november 2010.

[GLMS13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.

[Hol04] Gerard J. Holzmann. *The Spin Model Checker : Primer and Reference Manual*. Addison-Wesley, 2004.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of the European Conference on Object-Oriented Programming*, ECOOP '91, pages 133–147. Springer-Verlag, 1991.

[KS98] Ajay D. Kshemkalyani and Mukesh Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2):91–111, 1998.

[KS11] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, March 2011.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam02] Leslie Lamport. *Specifying Systems*. Addison Wesley, 2002.

[Lam09] Leslie Lamport. The PlusCal algorithm language. In Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing*, volume 5684 of *Lecture Notes in Computer Science*, pages 36–60. Springer, 2009.

[LFS+11] Xitong Li, Yushun Fan, Q. Z. Sheng, Z. Maamar, and Hongwei Zhu. A Petri net approach to analyzing behavioral compatibility and similarity of web services. *IEEE Transactions on Systems, Man and Cybernetics*, 41(3):510–521, May 2011.

[LW11] Niels Lohmann and Karsten Wolf. Decidability results for choreography realization. In *9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 92–107. Springer-Verlag, 2011.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[Mar03] Axel Martens. On compatibility of web services. *Petri Net Newsletter*, pages 12–20, 2003.

[Mat89] Friedemann Mattern. Virtual time and global state in distributed systems. In M. Cosnard, Y. Robert, P. Quinton, and M. Raynal, editors, *Int'l Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers, 1989.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[Mil82] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

[Mis83] Jayadev Misra. Detecting termination of distributed computations using markers. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 290–294. ACM, 1983.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems – Specification*. Springer-Verlag, 1992.

[OSB13] Meriem Ouederni, Gwen Salaün, and Tevfik Bultan. Compatibility checking for asynchronously communicating software. In *International Symposium on Formal Aspects of Component Software (FACS 2013)*, volume 8348 of *LNCS*, pages 310–328, 2013.

[Pal03] Catuscia Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[PBS89] Larry L. Peterson, Nick C. Buchholz, and Richard D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, 1989.

[PRS97] Ravi Prakash, Michel Raynal, and Mukesh Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41(2):190–204, March 1997.

[Ray10] Michel Raynal. *Communication and Agreement Abstractions for Fault-tolerant Asynchronous Distributed Systems*. Synthesis Lectures on Distributed Computing Theory. Morgan and Claypool Publishers, 2010.

[Ray13] Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013.

[RST91] Michel Raynal, André Schiper, and Sam Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39:343–350, October 1991.

[SCG+00] Mihaela Sighireanu, Claude Chaudet, Hubert Garavel, Marc Herbert, Radu Mateescu, and Bruno Vivien. LOTOS NT user manual, 2000.

[SM94] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, June 1994.

[Tel00] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, 2000.

[TFZ09] Wei Tan, Yushun Fan, and MengChu Zhou. A Petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, 6(1):94–106, 2009.