A Modular Framework for Verifying Versatile Distributed Systems

Florent Chevrou, Aurélie Hurault, Philippe Quéinnec

A Modular Framework for Verifying Versatile Distributed Systems

Florent Chevrou
Université de Toulouse, IRIT
ENSEEIHT – 2, rue Camichel
F31000 Toulouse, France
florent.chevrou@enseeiht.fr

Aurélie Hurault
Université de Toulouse, IRIT
ENSEEIHT – 2, rue Camichel
F31000 Toulouse, France
aurelie.hurault@enseeiht.fr

Philippe Quéinnec
Université de Toulouse, IRIT
ENSEEIHT – 2, rue Camichel
F31000 Toulouse, France
philippe.queinnec@enseeiht.fr

***Abstract*—Putting independent components together is a common design practice of distributed systems. Besides, there exists a wide range of interaction protocols that dictate how these components interact, which impacts their compatibility. However, the communication model itself always consists in a monolithic description of the rules and properties of the communication. In this paper, we propose a mechanized framework for the compatibility checking of compositions of peers where the interaction protocol can be fine tuned through assembly of individual properties on the communication. These include whether the communication is point-to-point or multicast, which ordering-policies are to be applied, applicative priorities, bounds on the number of messages in transit, and so on. Among these properties, we focus on a generic description of multicast communication that encompasses point-to-point and one-to-all communication as special cases. Eventually we provide theoretical views on the relations between ordering-policies through the lenses of multicast communication.**

***Index Terms*—Distributed systems; asynchronous communication; multicast; compatibility checking; TLA$^+$**

I. Introduction

Distributed systems are a composition of individual components, the peers, that exchange messages and work towards a common goal. Their interactions are governed by a protocol, or communication model, that specifies whether or not the emission or the reception of a message is possible. For example, synchronous communication dictates that a message shall be sent and received at the same time (rendez-vous). In asynchronous communication, though, which this paper focuses on, the emission and the reception of a message do not happen simultaneously: the two events occur with a delay. This results in many possible interleavings of the communication events, some of which might jeopardize the compatibility or the correction of a composition of peers unless specific properties on the communication are met. Such properties include whether the communication is point-to-point or multicast, numerous message-ordering policies that state some messages have to be delivered in their emission order, bounds on the number of messages in transit, and applicative priorities ensuring that some messages or recipients have precedence over others. Any conjunction of these properties is a unique communication model. Yet, existing verification frameworks consider the interaction protocol to be an indivisible entity that may be, at best, parameterized (e.g. capacity of queues) or entirely substituted by another.

In this paper, we describe an extensible framework where the communication model is any desired conjunction of communication properties we call "micromodels". It allows to verify with TLC, the TLA$^+$ model checker, properties on distributed systems depending on the combination of micromodels. Besides, we allow for different combinations to apply on different parts of the distributed system: for instance multicast causally ordered communication on the large scale but point-to-point capped FIFO ordered communication on a specific subsystem. Each micromodel is a transition system specified in TLA$^+$ whose transitions account for an emission or a delivery of message and whose states may fit any convenient data structure, no matter how the rest of the communication is described. For instance, a simple specification of the micromodel corresponding to the property "there are at most n messages in transit" is a set in which a message is added after an emission, removed after a reception, and that prevents any further emissions when it contains n messages. As an example, it may coexist with a micromodel that enforces a message delivery order using queues. The product of such an overall communication model and a composition of peers specified in TLA$^+$ constitutes the system to verify.

The presented framework handles both point-to-point (a message is delivered to one peer) and multicast (a message has several receivers) communication. Another contribution of this paper is a generic specification (in one single micromodel) of multicast communication that encompasses point-to-point and one-to-all communication as special cases. It relies on a notion called "interest" we motivate and describe.

The outline of this paper follows: Section II provides a brief introduction to the TLA$^+$ specification language, Section III presents the overall design of our verification framework and the modular design of communication models, Section IV details a universal micromodel of communication for both the point-to-point and multicast paradigms, Section V studies the relations between message-ordering multicast communication models, Section VI illustrates a use case of the verification tool with an example, Section VII explores related work, and eventually Section VIII sums this work up and paves the way for further developments.

II. TLA$^+$ Specification Language

TLA$^+$ [12] 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. TLA$^+$ allows to specify symbolic transition systems with variables and *actions*. An action is a transition predicate between a state and a successor state. It is an arbitrary first-order predicate with quantifiers, set and arithmetic operators, and functions. In an action, x denotes the value of a variable x in the origin state, and x' denotes its value in the next state. A specification of a system is usually a disjunction of actions. Fairness, usually expressed as a conjunction of weak or strong fairness on actions, or more generally as an LTL property, ensures progression. The TLA$^+$ toolbox contains the TLC model checker, the TLAPS proof assistant, and various tools such as a translator for the PlusCal Algorithm Language [13] into a TLA$^+$ specification.

III. Overview of the Verification Framework

The verification framework involves several independent TLA$^+$ modules that are connected during the verification process carried out by model checking using TLC. The key feature is a strict separation of concerns between the specification of the peers and the specification of the communication properties. The distributed system consists of the product of two transition systems: the composition of peers and the communication model which are both labeled by localized communication events.

A. Specification of a composition of peers

The specification of a composition of peers is a TLA$^+$ module that describes the state of each peer in the distributed system and specify their behavior according to transition predicates (actions). The module parameters the desired layout of micromodels of communication and instantiates the resulting communication model which then enables the peers to interact and exchange information. The beginning of Figure 6 gives an example set up of a communication model where a communication model COM is instantiated according to a layout of micromodels described in $COMMODELS$. In this example, one instance of a micromodel corresponds to one channel but it could be associated to any subset of the set of channels $CHANNELS$.

There is no restriction on the design of the specification of the composition. The actions in the composition may consist of a conjunction of an action from the instantiated communication model and a local state change. In practice, the state of the composition is usually a vector of every peer's state and the actions are localized. During an action in the system, the state of a peer evolves either spontaneously or alongside an action from the instantiated communication model. The available communication actions in a specification of a composition follow.

a) Send: $send(sender, receivers, channel, data)$ is enabled when the emission of a message by $peer$ on channel $channel$ is possible. We use the channel as an indirection on the notion of destination peer (point-to-point) or destination group (multicast). Besides, it makes it possible to specify systems where channels are not statically associated to a given sender or given group of receivers. $receivers$ restricts the set of possible receivers for this message: it is usually the set of all peers since channels dynamically account for the destination or destination group but it may be used to narrow a possible set of receivers down, send a message to an explicit destination, or to optimize the state space during model checking. Eventually, the payload of the message is $data$ without restriction on its type which can be adapted on a case-by-case basis. This payload is retrieved at delivery.

b) Receive: $receive(receiver, channel, data)$ is enabled when the reception of a message by $receiver$ on channel $channel$ that contains $data$ is possible. We assume peers cannot prevent a delivery based on the content $data$ of the message: the communication model imposes the message to be received and the content is only available afterwards. Therefore, in practice, a receive action in the specification of a composition has the form $\exists data \in DATATYPE : receive(\_, \_, data) \wedge P(data)$ where $P(data)$ is a transition predicate that covers all the possible values of $data$ in $DATATYPE$. This means that the next state of the receiver may depend on data but the enabledness of the reception itself is independent of this value.

c) Ignore: $ignore(peer, channels)$ is always enabled. It states that $peer$ does not expect to receive messages from the channels in $channels$ anymore. This cannot be reverted. The channels a peer has not ignored is called the *interest* of this peer. This information is crucial to the specification of some communication properties including multicast communication as detailed later in Section IV-B.

B. Specification of a communication model

1) Micromodels of communication: As previously stated, a communication model is a combination of communication properties we call micromodels. A micromodel has to answer the following two essential questions:

q1) When is the emission of a message, on a given channel, by a given peer, possible?

q2) When is the delivery of a message, on a given channel, to a given peer, possible?

In order to address these questions, the specification of a micromodel, a TLA$^+$ module, relies on its current state.

q3) Which information must the state carry?

Besides, a micromodel can be parameterized by constants in the module. For example, a micromodel corresponding to the property "the number of messages in transit is capped" has a parameter: the bound, and the state is the set of messages in transit. An emission requires the cardinality of this set not to exceed the limit and a delivery is always possible. This last point may seem odd, note though that the only purpose of this

micromodel is to limit the number of messages in transit. The basics of the communication such as "a message must have been sent before it is delivered" are part of another micromodel involved alongside. Micromodels are complementary with minimum overlap.

The remaining questions are then:

q4) What is the initial state?
q5) How does the state evolve after an emission?
q6) How does the state evolve after a delivery?
q7) How does the state evolve after some channels are ignored by a peer?

Since we aim at modeling both point-to-point and multicast communication, the answer to the last two questions is not trivial. Consider a micromodel that specifies either point-to-point or multicast communication and let us combine it with our example cap micromodel, characterized by a set of messages in transit. When performing a reception in this micromodel, the resulting state depends on the communication paradigm: the delivered message must be removed when the communication is point-to-point (the message is not in transit anymore) but the set may be left unchanged when the communication is multicast (the message remains in transit for further deliveries). We therefore distinguish two classes of micromodels: physical and non-physical. Physical micromodels specify when a message is removed from the communication model because it can no longer be received. Non-physical models specify predicates which control the sending and receiving of messages but are not concerned by the lifetime of a message. This information is fed to non-physical models by the physical models so they can evolve in a consistent way.

q8) Is the micromodel physical?

The specification of any micromodel, such as our running example (message cap) whose TLA$^+$ specification is in Figure 1, must answer each of the eight questions q1 to q8. The answer to q8 is a boolean $PhysicalMicromodel$; q1 and q2 are predicates $preSend$ and $preReceive$ that depend on the current state of the micromodel, the sender or receiver, the channel, and the data contained in the message; q3 is a type predicate $TypeInvariant$ depending on the current state s; q4 is the value $Init$ of the initial state; q5, q6, and q7 are the values $postSend$, $postReceive$, $postIgnore$ of the state after the operation. $postSend$ and $postReceive$ share the interface of $preSend$ and $preReceive$, $postIgnore$ depends on a peer and set of channels to ignore. Additionally, in the specification of non-physical micromodels, $postReceive$ has an additional boolean parameter $remove$ stating whether the received message should be removed or kept in transit, and $postIgnore$ has a set $removedIds$ of messages to remove.

2) Assembly of a communication model: An actual communication model is a combination of instances of micromodels, each corresponding to a subset of channels of the system. The following details an example communication model whose structure is summed up and illustrated in Figure 2. For instance, it is possible to state that, among channels a, b, c, d, e, and f, the communication has the property of a given micromodel on channels a, b, and c (say a message ordering property) and that another property (hence another instance of a micromodel such as the message cap micromodel) is associated to channels c and d. Overlaps are possible: communication on channel c has both the message ordering and the message cap properties. A micromodel can also be instantiated more than once: for example the message ordering micromodel can be instantiated again on channels e and f (i.e. micromodels 1 and 3 are two distinct instances of the same micromodel in the figure) which would mean messages on e and f are ordered, messages on a, b, and c, are ordered, but there is no guarantee on the ordering of a message of the first group and a message of the second group. As stated earlier, a physical micromodel dictates when a message no longer exists in the communication model (e.g. after the first delivery if the physical micromodel is point-to-point communication) and the information is used by the non-physical micromodels to update their local state. This implies that the sets of channels of physical micromodels must not overlap. Otherwise, two physical micromodels could disagree on whether to remove a message on a shared channel. However, the restriction does not apply to non-physical micromodels: the sets of channels may overlap or extend beyond the domains of physical micromodels. Given a communication model that is part point-to-point, part multicast, it is possible to limit the number of messages in transit on the whole communication model with a message cap instance that encompasses the domains of both the point-to-point and multicast physical micromodels.

3) Interlinking of the micromodels: The TLA$^+$ module that exposes the three communication operations available for the specification of compositions of peers is called the "multi-

MODULE $message\_cap$

EXTENDS $Naturals$, $FiniteSets$
CONSTANTS ID, $PEERS$, $CHANNEL$, $DATATYPE$,
$BOUND$ Maximum nb of messages in transit
$PhysicalMicromodel \triangleq$ FALSE $q8$

The state consists of one field: the ids of the messages in transit.
$TypeInvariant(s) \triangleq s \in [idInTransit : \text{SUBSET } ID]$ $q3$
$Init \triangleq [idInTransit \mapsto \{\}]$ $q4$

$usedIds(s) \triangleq s.idInTransit$

$preSend(s, id, from, to, channel, data) \triangleq$ $q1$
$Cardinality(s.idInTransit) < BOUND$
$postSend(s, id, from, to, channel, data) \triangleq$ $q5$
$[s \text{ EXCEPT } !.idInTransit = s.idInTransit \cup \{id\}]$

$preReceive(s, id, to, channel, data) \triangleq$ TRUE $q2$
$postReceive(s, id, to, channel, data, remove) \triangleq$ $q6$
IF $remove$ THEN $[s \text{ EXCEPT } !.idInTransit = @ \setminus \{id\}]$ ELSE s

$postIgnore(s, peer, chan\_set, removedIds) \triangleq$ $q7$
$[s \text{ EXCEPT } !.idInTransit = s.idInTransit \setminus removedIds]$

Fig. 1. TLA$^+$ module of a parameterized micromodel that caps the number of messages in transit. The annotations q1 to q8 indicate the answers to the questions a micromodel has to address.

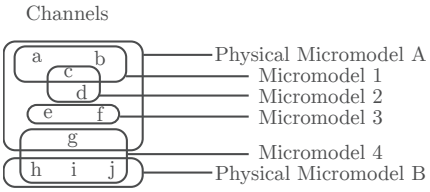

Fig. 2. A communication model built as a combination of micromodels. Each channel is associated to a unique physical micromodel.

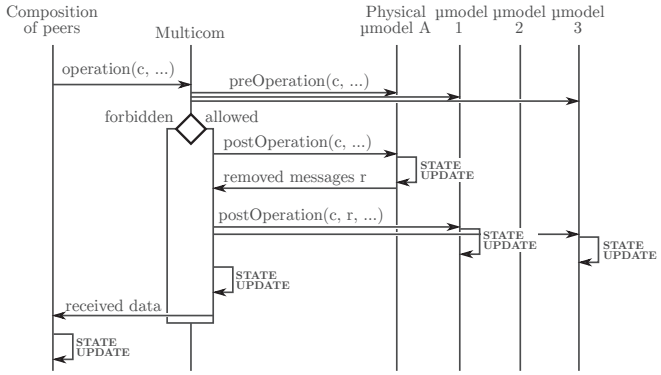

Fig. 3. Illustration of the dispatcher role of the multicom. An operation on channel c is initiated by a peer of a composition. It corresponds to a unique atomic TLA$^+$ action. The communication model is described in Figure 2: for channel c, it does not involve micromodel 2. The conjunction of the guards on the operation determines whether the operation is possible. If so, it is applied on the physical micromodel first and then on the others with knowledge of the removed messages.

com". It is instantiated in the specification of the composition of peers with a parameter: the specification of the communication model (see $COMMODELS$ in Figure 6). The multicom is a dispatcher that gathers the local states of the micromodels, checks whether an operation is possible (using the $pre\cdots$ predicates), and how the local states evolve (using the $post\cdots$ values). The multicom also generates and manages the message identifiers: a message has the same identifier across all the micromodels which makes it possible to maintain coherence. When an operation is to be performed, say a reception on channel c, the conjunction of all the $preReceive$ predicates of micromodels associated to c determines whether the reception is possible. If so, the new state of the physical micromodel of c is computed. By comparing it to the former state, the set of messages identifiers that are no longer in use (i.e. removed messages) is computed. It is provided to the non-physical micromodels whose state is updated afterwards. Figure 3 is a sequence diagram that gives insight into the process. Note that it is purely illustrative: it actually corresponds to a unique atomic TLA$^+$ action, that is a transition predicate involving the conjunctions of the micromodel-specific predicates.

IV. A PHYSICAL MICROMODEL FOR MULTICAST COMMUNICATION

A physical micromodel for asynchronous point-to-point communication can be modeled as a set of messages in transit, initially empty: the *network*. Sending a message is always enabled, by adding it to the network. Delivering a message requires it to be in the network and removes it. Obviously, a message is delivered at most once. In order to describe multicast communication which allows multiple deliveries of a message (at most one per peer), the lifespan of a message in transit must be subtly extended.

A. Lifespan of messages in transit

1) Sending the messages over and over: A simple solution would send the message again once it has been received so it can be received another time by another peer. There are two problems. This solution does not specify when to stop sending messages again. Second, when considering message-ordered communication where the order of the emissions matters (e.g. messages must be received in their emission order), sending a message again might modify the ordering. For instance, send m_1 followed by m_2, then deliver m_1. The semantics of this solution implies that m_1 is put back in the network and the new ordering is $m_2 \cdot m_1$ instead of $m_1 \cdot m_2$ the actual order of the multicast emission.

2) Never removing the messages from the network: Were the messages to remain in the network forever, they could be received as many times as needed. Once again however, this might conflict with some ordering policies. Assume that messages *must be* received in their emission order, that is to say the network can be viewed as a global queue, and consider two messages in transit. Even after all the peers have received the first message, since it remains in transit forever, none of them will ever receive the second (not first in queue) and the system will deadlock.

3) Removing a message from the network once delivered to all the peers: The previous issue is overcome by removing a message from the network after it has been delivered to every peer. Still, this means that all the peers must be ready to receive all the messages in order not to block the system. This requirement is too strong to allow for the verification of interesting and realistic systems: the specification of a peer should not depend on the noise in the environment it takes part in.

4) Removing after delivering to the relevant peers only: A compromise is to remove a message from the network as soon as it has been delivered to the peers involved in that message exchange while ignoring the others. This is reminiscent of [6] which specifies point-to-point message-ordering policies where delivery blockages arising from independent and irrelevant exchanges in the system are ignored.

B. Channels and interest as an indirection on destination groups

In order not to impose the delivery of a message that a peer has nothing to do with, and never will, we rely on the concept of interest. A peer is interested in some channels only: it expects messages on these channels. Over time, the peer may lose interest in some or all of them: either the expected deliveries have occurred or the peer has ruled out the possibility of ever receiving the messages. Action $ignore$

of the communication model allows a peer to lose interest in a given set of channels, as described in the previous section. The interest of the peers is part of the state of the multicast micromodel. The most sensible behavior would be to remove a message from the network as soon as the last peer interested in the channel of this message ignores it. However, a more generic approach is only a few tweaks away from this main rule.

C. A generic description for point-to-point, multicast, and one-to-all communication

The proposed operational specification of multicast communication is adapted to become generic and encompass, in particular, point-to-point communication. Consider two parameters of the communication denoted MIN and MAX.

- MIN is the minimal number of times a message must be received before it is removed from the network.
- MAX is the maximal number of times a message can be received before it is removed from the network regardless of the interest.

Let N denote the number of peers in the system. Up until now, we have described *multicast(0,N)* communication: a message is removed from the network when the corresponding channel does not interest any peer.

Point-to-point communication corresponds to *multicast(1,1)*. Indeed, a message must be received at least once before it can be removed from the network and must not be received more than once. This means it is immediately removed from the network following the first reception, never before. Similarly, *multicast(1,N)* corresponds to multicast communication were at least one peer must receive a message before it is removed, and *multicast(N,N)* models one-to-all communication where a message must be received by all the peers (including the sender) before it is removed from the network, regardless of the interest. MIN and MAX can also take any other value between 0 and N.

Figure 4 illustrates the differences between *multicast(0,N)*, *multicast(1,1)*, and *multicast(N,N)* with a common example scenario involving a global message-ordering policy (the network consists of a global common queue of messages). It shows the possible constraints and deadlocks that arise from combining two micromodels: a variant of multicast, and the global ordering policy.

The complete specification of the proposed generic micromodel is presented in Figure 5 and consists of two state variables $network$ and $interest$. The first one is a set of messages in transit which expands after each new emission; the second one contains, for each peer, the set of channels that it has not ignored (i.e. its interest). A message is composed of metadata including its unique identifier provided by the multicom, the sender, channel, and a set $receivedBy$ of peers it has already been delivered to in order to prevent multiple deliveries to the same peer (see $preReceive$). After a delivery (see $postReceive$), the receiver is added to the message's $receivedBy$ set but the message remains in the $network$ unless MAX receptions have occurred (after the first delivery in point-to-point, i.e. *multicast(1,1)*). When channels are ignored by a peer (see $postIgnore$), the $interest$ is updated, and messages that no longer interest any peer are removed from the $network$ unless they have not been delivered at least MIN times yet.

	$interest$			$network$		
Operation	p_1	p_2	p_3	*(0,N)*	*(1,1)*	*(N,N)*
	$\{a,b\}$	$\{a,b\}$	$\{a,b\}$	$\emptyset$	$\emptyset$	$\emptyset$
p_1 i a	$\{b\}$	$\{a,b\}$	$\{a,b\}$	$\emptyset$	$\emptyset$	$\emptyset$
p_1 ! a	$\{b\}$	$\{a,b\}$	$\{a,b\}$	a	a	a
p_1 ! b	$\{b\}$	$\{a,b\}$	$\{a,b\}$	$a \cdot b$	$a \cdot b$	$a \cdot b$
p_2 ? a	$\{b\}$	$\{a,b\}$	$\{a,b\}$	$a \cdot b$	b	$a \cdot b$
p_2 i a	$\{b\}$	$\{b\}$	$\{a,b\}$	$a \cdot b$	b	$a \cdot b$
p_3 ? a	$\{b\}$	$\{b\}$	$\{a,b\}$	$a \cdot b$	$\perp$[1]	$a \cdot b$
p_3 i a	$\{b\}$	$\{b\}$	$\{b\}$	b		$a \cdot b$
p_1 ? b	$\{b\}$	$\{b\}$	$\{b\}$	b		$\perp$[2]

[1] The message is not in the network anymore ($MAX = 1$).
[2] The message on a is still in the network ($MAX = N$) and must be received first according to the current ordering policy.

Fig. 4. Evolution of the state of the communication according to different instances of *multicast(*,*)* with global message-ordering, channels a and b, and $N = 3$ peers $(p_i)_{i \in 1..N}$. The network is represented by a queue. ! means "send", ? means "receive", i means "ignore".

V. Message-ordering properties

We provide non-physical micromodels for a large set of message-ordering policies. A detailed description, both axiomatic and operational, of classic point-to-point communication models is found in [6]. They include the following:

- *RSC* Realizable with Synchronous Communication [4], [10]. The emission of a message is immediately followed by its delivery (viewed atomically, it corresponds to synchronous communication).
- *FIFO n–n* Messages are globally ordered and are delivered in their emission order.
- *FIFO 1–n* Messages sent from a same peer are delivered in their emission order.
- *FIFO n–1* On a given peer, messages are received in their absolute emission order.
- *FIFO 1–1* Messages between a couple of peers are delivered in their emission order. Messages from/to different peers are independently delivered.
- *causal* Messages are delivered according to the causality of their emission [11]. 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.

The communication models in [6] are standalone and rely on message histories. By stripping away the management of the lifespan of messages in transit, we obtain specifications of their ordering policies that follow the previous conventions as pluggable and multicast-ready micromodels that make use of the concept of interest.

Totally-ordered multicast: Some distributed systems feature duplicated peers that are supposed to serve the same purpose and make the overall system more robust. A message that

```
MODULE multicast
EXTENDS Naturals, FiniteSets
CONSTANTS ID, PEERS, CHANNEL, DATATYPE, MIN, MAX
PhysicalMicromodel ≜ TRUE

LOCAL SendingSubsets ≜ (SUBSET PEERS) \ {}
LOCAL Message ≜ [
  id : ID,   Message identifier
  from : PEERS,   Sender
  to : SendingSubsets,   Possible receivers
  channel : CHANNEL,   Channel
  data : DATATYPE,   Payload
  receivedBy : SUBSET PEERS]   Peers it has already been delivered to
LOCAL Network ≜ SUBSET Message
LOCAL Interest ≜ [PEERS → SUBSET CHANNEL]

TypeInvariant(s) ≜ s ∈ [network : Network, interest : Interest]
Init ≜ [network ↦ {}, interest ↦ [peer ∈ PEERS ↦ CHANNEL]]
TransitingMessages(s) ≜ s.network ≠ {}
usedIds(s) ≜ {m.id : m ∈ s.network}

postIgnore(s, peer, chan_set) ≜
  LET new_peer_interest ≜ s.interest[peer] \ chan_set IN
    [s EXCEPT
      !.interest = [@ EXCEPT ![peer] = new_peer_interest],
      !.network = {m ∈ @ :
        ∨ m.channel ∈ new_peer_interest
        ∨ Cardinality(m.receivedBy) < MIN   not received enough
        ∨ ∃ p ∈ PEERS \ {peer} :   another is still interested
            m.channel ∈ s.interest[p]}]

  Emission: the message is added to the nework
preSend(s, id, from, to, channel, data) ≜ TRUE
postSend(s, id, from, to, channel, data) ≜
  LET related_messages ≜ {m ∈ s.network : m.from = from} IN
    [s EXCEPT !.network = @ ∪ {[id ↦ id, from ↦ from,
    to ↦ to, channel ↦ channel, data ↦ data, receivedBy ↦ {}]}]

preReceive(s, id, to, channel, data) ≜
  ∃ m ∈ s.network :
    ∧ m.id = id   The metadata of a message in transit match.
    ∧ to ∈ m.to
    ∧ m.channel = channel
    ∧ m.data = data
    ∧ to ∉ m.receivedBy   The peer has not received it yet.

postReceive(s, id, to, channel, data) ≜
  LET m ≜ (CHOOSE x ∈ s.network : x.id = id) IN
    The message has its receivedBy set updated first.
    LET network_preupdate ≜
      (s.network \ {m})
      ∪ {[m EXCEPT !.receivedBy = @ ∪ {to}]} IN
    The message is actually removed if it was the MAXth reception.
      [s EXCEPT !.network =
        {m2 ∈ network_preupdate :
          Cardinality(m2.receivedBy) < MAX}]
```

Fig. 5. TLA$^+$ specification of the generic multicast physical micromodel. The parameters MIN and MAX make it possible to use different instances of this module to model multicast communication, one-to-all communication, point-to-point communication, or in-between variants.

would be sent to a single peer in point-to-point communication is sent, in multicast communication, to all the duplicates. In such cases, it is interesting to guarantee that the messages are delivered in the same order to all the duplicates. This way, the receptions may be viewed as atomic, as if the duplicates where abstracted by a single peer that receives the message in question. This property is called totally ordered multicast and is independent from other ordering policies. As for now, we do not provide a micromodel of totally-ordered multicast communication. However, in the following, we identify ways to enforce the property using existing micromodels.

There exists a hierarchy of message-ordering policies with point-to-point communication. Consider a set of messages M and a set of peers P, let $E \triangleq \{s(p,m) \mid p \in P \wedge m \in M\} \cup \{r(p,m) \mid p \in P \wedge m \in M\}$ be the set of communication events: the disjoint union of the set of send and receive events. Each communication model is characterized by the set of sequences of events (called executions) it allows to unfold. For instance, the set of executions of *FIFO n–n* contains all the executions $(\sigma_i)_{i \in 1..N}$ (where $N \in \mathbb{N} \cup \infty$) such that $\forall m_1, m_2 \in M : \forall p_1, p_2, p_1', p_2' \in P : \forall i, j, i', j' \in 1..N : \sigma_i = r(p_1, m_1) \wedge \sigma_j = r(p_2, m_2) \wedge \sigma_{i'} = s(p_1', m_1') \wedge \sigma_{j'} = s(p_2', m_2') \Rightarrow \big((i < j) \Leftrightarrow (i' < j')\big)$. This means that if a reception happens before another, the two emissions of the messages must have happened in the same order. A communication model is stricter than another when the executions set of the former is included in the set of the later. Existing results with point-to-point communication reveal the following hierarchy from the strictest model to the most liberal:

Point-to-point communication:

- *RSC* $\rightarrow$ *FIFO n–n* $\rightarrow$ *FIFO n–1* $\rightarrow$ *causal* $\rightarrow$ *FIFO 1–1* $\rightarrow$ *no-ordering*
- *RSC* $\rightarrow$ *FIFO n–n* $\rightarrow$ *FIFO 1–n* $\rightarrow$ *causal* $\rightarrow$ *FIFO 1–1* $\rightarrow$ *no-ordering*

We have extended these results to multicast communication. As it turns out, the point-to-point hierarchy holds apart from the 1-n $\rightarrow$ causal link that no longer stands but more importantly, we have been able to prove that *FIFO n–1* communication suffices to guarantee totally-ordered multicast. The proof is detailed in [5]. The resulting hierarchy is the following:

Multicast communication:

- *RSC* $\rightarrow$ *FIFO n–n* $\rightarrow$ *FIFO n–1* $\rightarrow$ *causal* $\rightarrow$ *FIFO 1–1* $\rightarrow$ *no-ordering*
- *RSC* $\rightarrow$ *FIFO n–n* $\rightarrow$ *FIFO 1–n* $\nrightarrow$ *causal*
- Additionally: *FIFO n–1* $\rightarrow$ *totally-ordered multicast*

Although we do not propose a micromodel of totally-ordered multicast communication, we have been able to identify the more liberal and already available micromodel that provides the property. In the example described in the following section, we make use of this knowledge in a use case involving both *multicast(0,N)* and *FIFO n–1*.

VI. Example

Let us consider a conference reviewing system. The peers are the authors, the chairs of the program committee and

the reviewers. Authors send their papers to all the PC chairs (multicast to the chairs). Each chairperson attributes a paper number and takes responsibility for a part of the papers, based on this number. In order that all chairs attribute the same number to a given paper, and without internal coordination between the PC chairs, the authors use a totally ordered multicast so that the papers are delivered in the same order to all the chairs (fifo n-1 ordering model). After the deadline has passed, the chairs reject new submissions (point-to-point communication to the author). After the deadline, each chair independently sends its papers to some of the reviewers (bounded multicast to the reviewers), waits for the reviews (multicast from the reviewers to the chairs), and sends the acceptance result to the author (point-to-point). The system must ensure that it does not deadlock and that every author eventually receives an answer (either rejection for a late paper, or acceptance result if reviewed). This system exposes both strict ordering constraints (submissions sent to the chairs), and high interleaving (each reviewer is independently handling the papers it has received). The system has been described in the PlusCal Algorithm Language, which is translated by the TLA$^+$ tools to a TLA$^+$ specification, which can then be checked with the TLC model checker. An excerpt[1] is given in Figure 6. Results have shown that the message cap on the number of messages in transit is instrumental to avoid state explosion as it ensures that messages are not delayed for too long. During the development of the system, several bugs were found. For instance, the logic to split the papers among the chairs was faulty with an odd number of chairs (e.g. one…) and some authors were never receiving their acceptance result; in some cases, the same paper was sent twice to the reviewers, and an unfortunate (but legal) interleaving in the reception of the reviews led to two acceptance messages to the same author. This system, albeit simple, already experiences enough communication interactions to warrant formal verification.

```
MODULE reviewing
CONSTANT NbAuthors, NbChairs, NbReviewers,
         NbMinReviews, NbMaxReviews, Capacity

CHANNELS ≜ {"submission", "paper", "review", "acceptation"}
COMMODELS ≜ {
   [name ↦ "multicast", params ↦ [chan ↦ {"submission"},
                 min ↦ 1, max ↦ NbChairs]],
   [name ↦ "multicast", params ↦ [chan ↦ {"paper"},
              min ↦ NbMinReviews, max ↦ NbMaxReviews]]
   [name ↦ "multicast", params ↦ [chan ↦ {"review"},
                 min ↦ 1, max ↦ NbChairs]],
   [name ↦ "p2p", params ↦ [chan ↦ {"acceptation"}]],
   [name ↦ "fifon1", params ↦ [chan ↦ {"submission"}]],
   [name ↦ "message_cap", params ↦ [chan ↦ CHANNELS,
                          bound ↦ Capacity]]}

COM ≜ INSTANCE multicom WITH
          PEERS ← IdAuthors ∪ IdChairs ∪ IdReviewers,
          COM ← COMMODELS,
          CHANNEL ← CHANNELS
 ....
--algorithm reviewing
 ....
fair process Reviewer ∈ IdReviewers
variable
  readinglist = {} ;  – for each reviewer, the papers he has to review
begin
rl0:            – listen only on channel "paper"
  await ignore(self, CHANNELS \ {"paper"}) ;
rl1:
  while TRUE do
    either          – receive a paper to review
      await Cardinality(readinglist) ≤ 4 ;
      with paper ∈ IdPapers do
        await COM!receive(self, "paper", paper) ;
        readinglist := readinglist ∪ {paper} ;
      end with ;
    or              – send a review to the chairs
      with paper ∈ readinglist do
        await COM!send(self, IdChairs, "review", ⟨self, paper⟩) ;
        readinglist := readinglist \ {paper} ;
      end with ;
    end either ;
  end while ;
end process
end algorithm
```

Fig. 6. An excerpt of the conference reviewing system. COMMODELS specifies the properties of the channels (e.g. `review` is a multicast 1-NbChairs channel). The peers (`processes` in PlusCal language) are the authors, the chairs and the reviewers. The reviewers have two actions: they can either receive a message on channel `paper` or send a message on channel `review`. The chairs and the authors (not shown here) have respectively five and two actions.

VII. Related Work

Tel's textbook [17] 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) point-to-point communication whereas we explicitly describe the communication subsystem with a transition system, consider several communication properties, including multicast and message-ordering policies, compare them, and offer a mechanized framework for checking compositions of peers.

Promela (Process Meta Language) [9] is used to specify state transition systems that may describe distributed systems and asynchronous interactions. The associated model checker, SPIN, performs efficiently on these specifications. However, Promela only provides FIFO message channels to model the communication whereas our work requires an approach that encompasses the variety of asynchronous communication properties and the deriving communication models.

Micro-protocols have been used in Horus [16] and Ensemble [15]. The developer arranges a stack of micro-protocols to obtain precisely the desired properties. Each micro-protocol layer handles some small aspect of these properties. For instance, one layer might deal with message loss, one with encryption, one with group membership, and another one with

[1]The complete files, of the example and of the used communication models, are available on http://vacs.enseeiht.fr/4pad2018/.

multicast ordering. One notable point of Ensemble was the use of NuPrl for provably rewriting the stack and generating optimized implementations [14], and of I/O automata for formalizing, specifying, and verifying the Ensemble implementation [8]. The main differences with our work is the hierarchical structure of the stack, and that the objectives of Horus and Ensemble was to provide efficient implementations of a group-communication infrastructure and was not concerned with the verification of the applications themselves.

Compatibility of services or software components has largely been studied, especially with regards to collaborations and choreographies. Usually the interaction model is fixed and global for all the interactions. The majority of the approaches consider synchronous communication (e.g. [3], [7]), even if a few works consider asynchronous communication with variations of FIFO ordering (e.g. [2], [1]). To the best of our knowledge, no work has considered multicast communication or composed communication models.

VIII. Conclusion

This paper proposes an approach to the verification of asynchronous distributed systems that considers the influence of each individual property of the communication medium on the peers of the system. The first contribution is a verification framework in TLA$^+$ that offers to build communication models by combining individual communication properties we call micromodels. It benefits from the TLA$^+$ tools: the model checker TLC and the proof assistant TLAPS. This allows to cover a wide range of possible asynchronous communication variants while existing verification tools usually stick to a few particular cases and seldom offer much control over the features of the communication medium. Each specification of a micromodel follows a simple yet generic template allowing for easy expansion. Among them, we distinguish physical micromodels that specify when a message is removed from the whole communication model, and non-physical micromodels that provide additional properties among message ordering, cap on the number of messages in transit, or applicative priorities. The second contribution is a physical micromodel that encompasses both point-to-point and multicast communication thanks to a notion called the interest. The interest is an indirection on the usual notion of destination group in multicast communication: a message is proposed for delivery as long as some peers are or may later be interested in receiving it. By tweaking this rule with two parameters MIN and MAX that respectively prevent or force the removal of a message from the communication model depending on the current number of deliveries, we describe the whole spectrum of multicast communication spanning from point-to-point to one-to-all communication.

Ongoing work aims at specifying a micromodel for totally-ordered multicast communication. This ordering policy happens not to integrate as easily as other classic ordering policies with the notion of interest. Knowledge on the future behavior of the peers is necessary to check whether a delivery violates the ordering. Further thinking is thus required. In the meantime, this paper identifies the weakest classic ordering policy that provides totally-ordered multicast in a ready-to-use micromodel. Considering fault models among message loss, duplication, or crash of a peer, is another perspective. More generally, we do not specify the behavior of classic ordering policies after a loss or duplication of a message, or any other failure. The two challenges involve adapting existing micromodels and studying how the very notion of fault models can be integrated in the framework: it may require small adaptations or deeper refactoring.

References

[1] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *39th Symposium on Principles of Programming Languages*, POPL '12, pages 191–202. ACM, 2012.

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

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

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

[5] Florent Chevrou. *Formalisation of Asynchronous Interactions*. Phd thesis, Institut National Polytechnique de Toulouse, Toulouse, France, November 2017.

[6] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Aspects of Computing*, 28(5):847–879, September 2016.

[7] 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.

[8] Jason J. Hickey, Nancy Lynch, and Robbert van Renesse. Specifications and proofs for Ensemble layers. In R. Cleaveland, editor, *Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, LNCS 1579, pages 119–133. Springer-Verlag, 1999.

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

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

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

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

[13] Leslie Lamport. The pluscal algorithm language. In Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing - ICTAC 2009, 6th International Colloquium*, volume 5684 of *Lecture Notes in Computer Science*, pages 36–60. Springer, 2009.

[14] Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason J. Hickey, Mark Hayden, Kenneth Birman, and Robert Constable. Building reliable, high-performance communication systems from components. In David Kotz and John Wilkes, editors, *17^{th} ACM Symposium on Operating Systems Principles (SOSP'99)*, volume 33(5) of *Operating Systems Review*, pages 80–92. ACM Press, December 1999.

[15] Robbert van Renesse, Kenneth P. Birman, Mark Hayden, Alexey Vaysburd, and David Karr. Building adaptative systems using Ensemble. *Software – Practice and Experience*, 28(9):963–979, August 1998.

[16] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A flexible group communications system. *Communications of the ACM*, 39(4):76–83, April 1996.

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