Electronic Communications of the EASST
Volume 072 (2015)

Proceedings of the
15th International Workshop on
Automated Verification of Critical Systems (AVoCS 2015)

Automated Verification of Asynchronous Communicating Systems with TLA$^+$

Florent Chevrou, Aurélie Hurault, Philippe Quéinnec

15 pages

Guest Editors: Gudmund Grov, Andrew Ireland
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Automated Verification of Asynchronous Communicating Systems with TLA$^+$

Florent Chevrou, Aurélie Hurault, Philippe Quéinnec

firstname.lastname@enseeiht.fr
IRIT – Université de Toulouse
2 rue Camichel
F-31000 Toulouse, France
http://www.irit.fr

Abstract:

Verifying the compatibility of communicating peers is a crucial issue in critical distributed systems. Unlike the synchronous world, the asynchronous world covers a wide range of message ordering paradigms (e.g. FIFO or causal) that are instrumental to the compatibility of peer compositions. We propose a framework that takes into account the variety of asynchronous communication models and compatibility properties. The notions of peer, communication model, system and compatibility criteria are formalized in TLA$^+$ to benefit from its verification tools. We present an implemented toolchain that generates TLA$^+$ specifications from the behavioral descriptions of peers and checks compatibility of the composition with respect to given communication models and compatibility criteria.

Keywords: aynchronous communication, peer composition, compatibility checking, TLA$^+$

1 Introduction

Building systems through assembling and coordinating off-the-shelf components is a thriving software production principle. The formal verification of the correctness of the composition of a set of peers is crucial to this approach when it comes to critical systems. In this setting, the interaction model can directly impact the properties of the global system. 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 algorithms. For instance, the Chandy-Lamport snapshot algorithm [CL85] requires that the communication between two processes is FIFO, and Misra's termination detection algorithm [Mis83] works with a ring containing each node once if the communication ensures causal delivery, but requires a cycle visiting all network edges if communication is only FIFO.

Although the question of characterizing the properties of a set of combined services has been extensively studied for quite a long time (e.g. [BZ83, LW11]), existing works are restricted, to the best of our knowledge, to a specific interaction model (either synchronous or asynchronous, or coupling via bounded buffers), to which their formalization and verification framework are dedicated. Moreover the diversity of asynchronous communication models is generally ignored.

We present a framework and a ready-to-use automated toolchain, based on TLA$^+$, that enables to check LTL properties on distributed systems. A system is the conjunction of peers and communications models (TLA$^+$ modules) that interact through channels. Unlike many existing approaches, explicit senders and receivers are not required which allows for a greater variety of specifications. Although several communication models and compatibility LTL properties are supplied, the framework can be extended at will with additional ones thanks to its modular structure and transition system base. These additions can be generic or fulfill system-specific needs (case-by-case adaptation). Finally, it integrates well with other tools: peer specification helpers, not part of the core framework, are provided in the presented toolchain.

The outline of this paper is the following. Section 2 introduces our views on asynchronous communication and the choices made to model interaction. Several classic communication models are then presented. They highlight the diversity of asynchronous communication. Section 3 presents the core framework and an automated toolchain where peers are specified with transition systems derived from CCS terms. Section 4 presents a use case example and a performance benchmark. Section 5 provides an overview of the conceptual background of this work and, eventually, the conclusion draws perspectives after summing up this work.

2 Asynchronous communication

2.1 Intuition

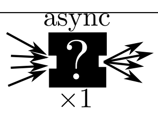
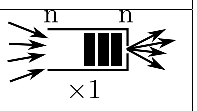
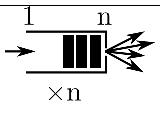
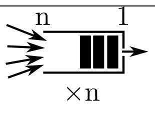
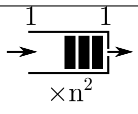
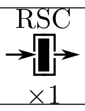
Communication consists in exchanging messages whose content is not relevant outside the scope of peers' internal behavior. Messages are sent on channels. Channels do not have explicit sender and receiver and are not limited to one sender/one receiver. They are nevertheless a point-to-point communication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

From the traditional distributed systems viewpoint, the communication medium controls the messages deliveries. Peers cannot impose a delivery order. However, a peer specifies which channels it listens to in order to prevent the delivery of a message it is not and never will be concerned about. For instance, a peer that only expects to receive a message from channel a followed by a message from channel b cannot impose the reception order (the communication model will have to if this is essential to compatibility), but it can specify it only listens to channels a and b, thus preventing the communication medium from imposing a message from another channel.

2.2 Communication Models

We describe seven asynchronous communication models in table 1 and provide instance implementations, often based on queues, to illustrate them. They enforce an order of message receptions in relation with the order of their emissions. For instance, in M_{n-1}, each peer has a unique mailbox which is FIFO-ordered. Messages are delivered to this peer in their absolute send ordering, whatever the sending peers are. This contrasts with M_{1-1} where a queue is present between every couple, and no order is imposed between two messages coming from different peers.

Table 1: Communication Models Description

Model	Specification	Intuitive Implementation	
M_{async}	Fully asynchronous. No order on message delivery is imposed.	A bag from which messages are non-deterministically retrieved.	
M_{n-n}	Global ordering. Messages are delivered in their send order.	A unique FIFO queue where all messages are put in and retrieved from.	
M_{1-n}	Messages from the same peer are delivered in their send order.	An output queue for each peer (outbox) from which messages are instantly retrieved.	
M_{n-1}	On a given peer, messages are received in their send order.	An input queue for each peer (mailboxes) where messages are instantly deposited.	
M_{1-1}	Messages between two designated peers are delivered in their send order.	A FIFO queue between each couple of peers.	
M_{causal}	Messages are delivered according to the causality of their emission [Lam78].	Using causal histories [SM94] or logical matrix clocks [RST91].	
M_{RSC}	Messages are immediately delivered after their send [CMT96].	A 1-slot unique buffer shared by all peers.	

Bounded Implementations Some variations also include the possibility to count and/or limit the number of in transit messages, locally (peer) or globally. A counter of in transit messages is updated at send and receive events. It is used in a threefold manner. First, an enforced limit models a bounded network where the emission of a message is not always enabled. Secondly, the state space is reduced, thus making its exploration quicker. Thirdly, for a correct finite system, the maximal value of the counter is the highest number of in transit messages.

Composite Models We also consider composite communication models made up of several other models. Each one of them manages communication on its own subset of channels. This makes it possible to ensure different ordering properties on these channel groups.

3 A Framework for the Verification of Asynchronously Communicating Peers

This section presents a framework aimed at checking compatibility properties over a composition of a set of peers and a communication model (possibly composite). Peers and communication models are both specified using transition systems. Interactions between them are represented by

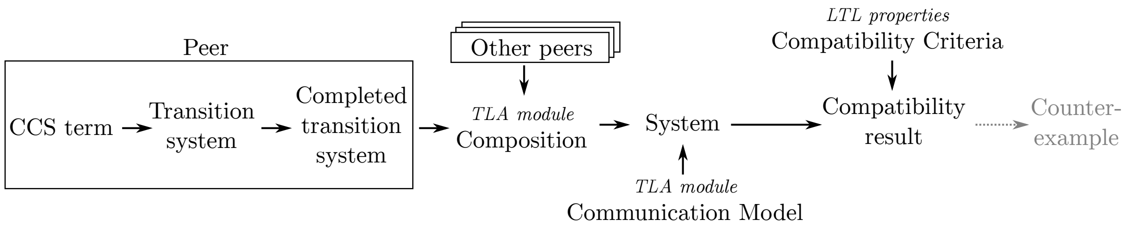


Figure 1: Main Steps Performed by the Framework

a synchronous product. These notions are translated into TLA$^+$ specifications where this product appears as a conjunction of actions. The choice of TLA$^+$ as the specification language arises from the high-level structures (such as sets, tuples and functions) it offers. This paves the way for evolved communication models implementations that for instance rely on nested message histories. The "actions as predicates" approach also eases the synchronous product operation.

Figure 1 provides an overview of the different implemented steps to perform the automatic verification of a composition. They are detailed in the following sections.

3.1 TLA$^+$ Specification Language

TLA$^+$ [Lam03] 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. System properties are specified using TLA which is a variant of linear temporal logic (LTL).

The dynamic behavior of a system is expressed as a transition system whose specification is usually written as $Init \wedge \Box[Next]_{vars} \wedge \mathscr{F}$, where $Init$ is a predicate specifying the initial states, $Next$ is the transition relation, usually expressed as a disjunction of *actions*, and $\mathscr{F}$ expresses fairness conditions. Weak fairness $\mathrm{WF}_v(A)$ means that either infinitely many A steps occur or A is infinitely often disabled. 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. UNCHANGED x means that $x' = x$.

Functions are primitive objects in TLA$^+$. The application of function f to an expression e is written as $f[e]$. $[x \in X \mapsto e]$ denotes the function that maps any $x \in X$ to e, and $[f \text{ EXCEPT } ![e_1] = e_2]$ is a function which is equal to f except at point e_1, where its value is replaced with e_2. Tuples (a.k.a. sequences) are functions with domain $1..n$. Tuples are written $\langle a_1, a_2, a_3 \rangle$. $\langle\rangle$ is the empty sequence. Modules are used to structure complex specifications. They can *extend* other modules, importing all their declarations and definitions, or be an *instantiation* of another module. $MI \triangleq \text{INSTANCE } M \text{ WITH } q_1 \leftarrow e_1, q_2 \leftarrow e_2 \ldots$ is an instantiation of M, where each symbols q_i is replaced by the expression e_i. $MI!x$ references x in the instantiated module.

3.2 System Model

A system is composed of a set of indexed peers $P_1, \ldots, P_n$ and a communication model M (possibly composite). They are specified using transition systems labelled by communication events.

Communication occurs when matching transitions in M and one of the P_i are synchronized and the P_j ($j \neq i$) stutter. Internal actions τ can occur without synchronization. Since delivery is stable in our models described in table 1, a minimal progress property (weak fairness in TLA$^+$) prevents infinite stuttering.

Channels are used instead of explicit sender and receiver. Thus, peer transitions are only characterized by the nature of the communication (send "!" or receive "?") and the concerned channel (e.g. $\xrightarrow{c!}$). Peer states are characterized by program counters in TLA$^+$ modules (e.g. figure 2). The state of a composition $P_1,\dots,P_n$ is an array of program counters which carries P_i's state at index i. As for M, a send (resp. receive) transition accounts for a peer that has sent (resp. received) a message. However, unlike peers, M also requires information about the identity of the peer concerned by the communication operation to guarantee interesting ordering properties. For instance, $\xrightarrow{3,c!}$ in M is to be synchronized with a $\xrightarrow{c!}$ transition in P_3. A TLA$^+$ module corresponding to a communication model is specified using state variables and transition predicates that implement the rules of message ordering. The notion of listened channels mentioned in 2.1 is also taken into account as additional information in the case of receive transitions, both in the peers and communication models. For instance, in M, $\xrightarrow{2,a?,\{a,b,e\}}$ accounts for the reception of a message on channel a by P_2 in a context where P_2 listens to channels a, b, and e. It is to be synchronized with a $\xrightarrow{a?,\{a,b,e\}}$ transition in P_2. Although the inner functioning of the peers is often irrelevant when it comes to checking the compatibility of a composition, modeling data passing can be of interest and easily handled by additional fields on the transitions. Such cases include lifting indeterminism, creating, or transmitting channels. Peers would have to be specified accordingly in a more complex manner than with basic program counters.

Figure 2 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 departs from state 14, depending on the reception channel.

3.3 Causal Commununication Model (M_{causal}) Implementation

As an example, figure 3 shows the TLA$^+$ module corresponding to an implementation of the causal communication model M_{causal}. The causal order [Lam78] is the weakest partial order which contains both the peer local order and the send-receive order. Message histories are used to keep track of this order. The state variables are net the set of in transit messages, and H the array composed of a history (message set) for each peer. A message consists of a channel, the sender id, and a snapshot of the sender's history at send[1]. The reception predicate requires that no in transit message (whose channel is listened to) appears in the history snapshot of the to-be-received message. Thus it garantees that messages are not received in an order that would violate the causality of their emission. The local history of each peer is updated in a way that describes the causal ordering: at send and receive, the message is added in the peer local history (see "$\cup\{message\}$" at α and "$\cup\{\langle c1, p1, h1\rangle\}$" at β); the link between send and receive is performed when merging the message's history to the receiver's local history (see "$\cup h1$" at β).

[1] Messages are unique because a peer history is strictly increasing.

MODULE *composition*

EXTENDS *Naturals*, *peermanagement*
CONSTANTS N
VARIABLES *net*
$Vars \triangleq \langle peers, net \rangle$
$Com \triangleq$ INSTANCE *causal* WITH $CHANNEL \leftarrow \{$"a", "b"$\}$

$Init \triangleq Com\,!\,Init \wedge peers = \langle 11, 14 \rangle$ 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: $a?.b?$

$t3(peer) \triangleq trans(peer, 14, 15) \wedge Com\,!\,receive(peer, \text{"a"}, \{\text{"b"}, \text{"a"}\})$ **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:** $state_{14} \xrightarrow{b?} state_{17}$

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

Figure 2: Generated TLA$^+$ Module: $\xrightarrow{a!}\ \xrightarrow{b!}$ Composed with
$$\begin{tikzcd} \bullet \arrow[r, "a?"] \arrow[dr, "b?"'] & \bullet \arrow[r, "b?"] & \bullet \\ & \bullet & \end{tikzcd}$$

3.4 Compatibility

We define two universal peer states: 0 the terminal state, and $\perp$ the faulty state. 0 characterizes a peer that has reached a point where the tasks it was supposed to perform are done. $\perp$ is 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 transition leads to 0, $\perp$, or another state is part of the peer specification. A compatibility property is given as an LTL formula. We denote s_i the state of peer i and the following predicates are defined:

$0_\forall$	$\triangleq$	$\forall i \in 1..n : s_i = 0$	peers are all in the terminal state
0_i	$\triangleq$	$s_i = 0$	termination of peer i
$\perp_\exists$	$\triangleq$	$\exists i \in 1..n : s_i = \perp$	an unexpected message has been delivered

The following compatibility properties are defined:

System Termination	The system always reaches a terminal state.	$System \models \Diamond\Box 0_\forall$
Peer Termination	Peer i always reaches a terminal state.	$System \models \Diamond\Box 0_i$
No faulty receptions	No unexpected reception ever occurs.	$System \models \Box\neg\perp_\exists$

No forever blocking communication At any time, at least one communication event is possible (except if terminated or after a faulty reception).
$System \models \Box(0_\forall \vee \perp_\exists \vee \text{ENABLED}(R))$ where R is the system transition relation.

In the figure 2 example: if we replace states 13 and 16 by 0, and 17 by $\perp$, M_{causal} makes it impossible to reach $\perp$ and the four mentioned compatibility properties hold.

```
MODULE causal
EXTENDS $Naturals, FiniteSets$
CONSTANTS $CHANNEL, N$   The channels managed by the model and the number of peers
VARIABLES $net, H$
$Init \triangleq \land net = \{\} \land H = [i \in 1..N \mapsto \{\}]$   Network and histories are initially empty

$send(peer, chan) \triangleq$   Emission from peer of a message on channel $chan$
    LET $message \triangleq \langle chan, peer, H[peer] \rangle$ IN   Message content: channel, sender, and sender current history
      $\land\ net' = net \cup \{message\}$   The message is added to the network
      $\land\ H' = [H$ EXCEPT $![peer] = H[peer] \cup \{message\}]$   $\alpha$ : This send is made part of the peer history

$receive(peer, chan, listened) \triangleq$   Reception, on peer, of a message on $chan$
    $\exists \langle c1, p1, h1 \rangle \in net :$   There is an in transit message such that
        $\land\ c1 = chan$   The channels match
        $\land\ \neg(\exists \langle c2, p2, h2 \rangle \in net : c2 \in listened \land \langle c2, p2, h2 \rangle \in h1)$   No in transit message of interest is in conflict
        $\land\ net' = net \setminus \langle c1, p1, h1 \rangle$   It is retrived from the in transit messages
        $\land\ H' = [H$ EXCEPT $![peer] = H[peer] \cup h1 \cup \{\langle c1, p1, h1 \rangle\}]$   $\beta$ : The peer history is updated
```

Figure 3: TLA$^+$ Module Associated to the Causal Communication Model

3.5 User Friendliness

Explicitly defining even quite simple peer transition systems can be cumbersome. One may want to step back and provide more abstract specifications. The proposed framework provides alternate ways to assist peers specification.

3.5.1 Peer Alternative Specification

A peer can alternatively be defined by a process specified with a CCS term. The peer transition system is derived from the CCS term using the standard CCS rules [Mil99, p.39] and excluding the synchronous communication rule. The translation from a CCS term to a transition system is achieved through the Edinburgh Concurrency Workbench [CPS93]. 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 the entire peers transitions. PlusCal specifications, for instance, could also offer practical alternatives to CCS and the explicit generation of transitions.

3.5.2 Faulty Reception Completion

The faulty receptions completion (FRC) consists in revealing the unexpected receptions in a peer and mark them as faulty by adding a corresponding transition toward $\perp$. It makes peers fit the intuitive viewpoint where the communication medium impose messages: if a peer is not interested in a channel at a given time, it will never be later.

For each state s where a receive transition exists, the future channels of s is the set of channels corresponding to possible future receptions. For each channel c in the future channels that is not already specified as an alternative choice in this state, such a choice is provided by a transition towards $\perp$ and labeled by $c?$: $s \xrightarrow{c?} \perp$. These are called *faulty receptions*.

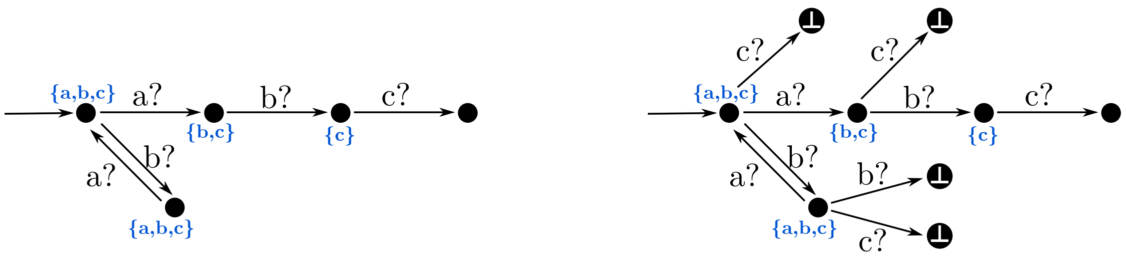


Figure 4: Faulty Receptions Completion Example

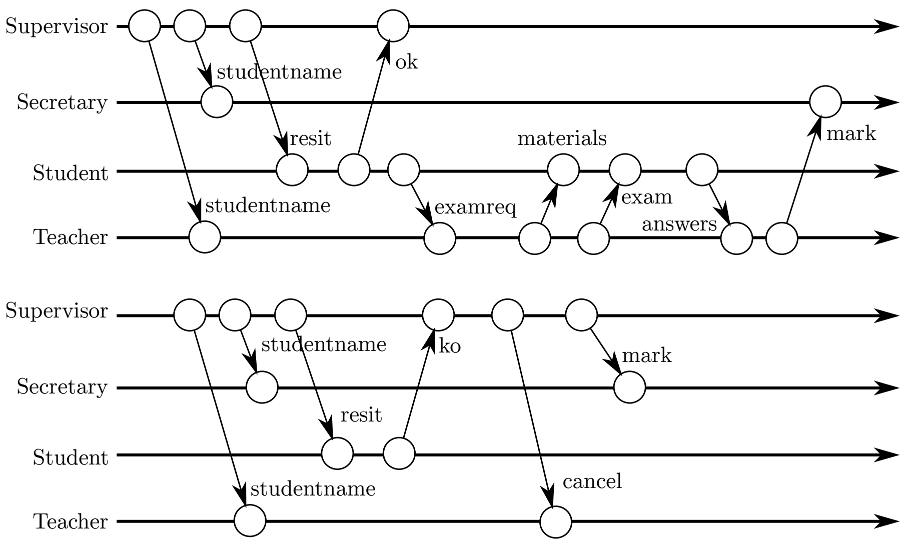


Figure 5: Expected Executions Examples

For instance, let us consider the peer represented on the left in figure 4. 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 $a! \cdot b! \cdot c! \cdot 0$ and M_{1-1}, the faulty receptions are impossible (because the send order must be 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} for example.

4 Experiments and Results

This section presents a concrete example which illustrates the interest of a diversity of asynchronous models, and some benchmark results which show the usability of the framework for larger systems.

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

Figure 6: Supervisor-Secretary-Student-Teacher Specification

4.1 Practical Example

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 and can resit, he 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 answers ok and asks the teacher for the exam. The teacher then sends the needed materials and then the exam, after which the student sends back his answers, then the teacher sends a mark to the secretary. If the student declines to resit, he informs the supervisor who sends a cancel message to the teacher and the former mark to the secretary. Sample executions are depicted in figure 5 and the system is specified in figure 6.

We consider the models defined in table 1 and the composite model M_{comp}:

$$
M_{comp} = \begin{cases} M_{causal} & \{studentname, resit, examreq, cancel, mark\} \\ M_{1-1} & \{materials, exam\} \\ M_{async} & \{ok, ko, answers\} \quad \text{(no constraint)} \end{cases}
$$

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, compatibility, especially termination of the secretary service, is not trivial. Consequently, in addition to the generic compatibility properties defined in 3.4, we also consider the termination of the secretary and we check if all messages have been received upon full termination.

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 $studentname$. Finally, the $materials$ and the $exam$ are sent in two separate messages and are not expected to be received in the reverse order by the student.

Figure 7 presents the results. It confirms that causality is needed to ensure compatibility of the composition but not required over the whole set of channels. The considered composite model is restrictive enough. In this example, with that composite communication model, model checking

	M_{n-n}	M_{1-n}	M_{n-1}	M_{causal}	M_{1-1}	M_{async}	M_{comp}
Termination	✔	✔	✔	✔	×	×	✔
Termination with an empty network	✔	✔	✔	✔	×	×	✔
Partial termination (secretary)	✔	✔	✔	✔	×	×	✔
No faulty receptions	✔	✔	✔	✔	×	×	✔
No forever blocking communication	✔	✔	✔	✔	×	×	✔

Figure 7: Compatibility Results

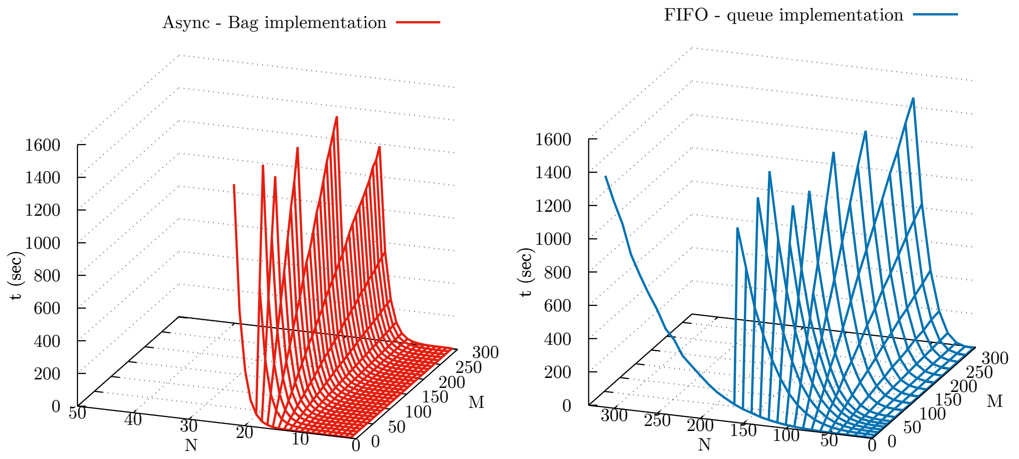


Figure 8: Runtime for the Studied System

generates 135 distinct states.

4.2 Benchmarking

Studied system We study the composition of two peers derived and completed from the following CCS terms: $(a_1! \cdot \ldots \cdot a_N!.b?)^M$ and $((a_1?\| \ldots \| a_N?) \cdot b!)^M$. It consists in transmitting M series of N messages (emitted in the same order and that can be received in any order) separated by a synchronization message. We check for termination and observe the number of generated states and runtime. This benchmark is relevant to study looping systems consisting in sections where ordering may be crucial, explicitly sequenced by synchronization points. Depending on the communication model, results are expected to vary. Indeed, without constraint (M_{async}), all the reception interleavings are possible, while other models like M_{1-1} impose a single path that corresponds to the send order. These are two extreme cases. We rely on a bag-based implementation for M_{async} and a sequence-based implementation for M_{1-1}. The tests ran on a machine with 2×4 cores Intel Xeon CPU E5-2690 v3 at $2.60\,GHz$ and $23\,GiB$ of RAM.

Results The results are presented in figures 8 and 9. They show that the number of states and runtime increase linearly with M the number of critical sequences. They increase exponentially when it comes to N because it accounts for the maximum number of in transit messages at a given time and all the possible receptions that have to be tried during model-checking. M corresponds

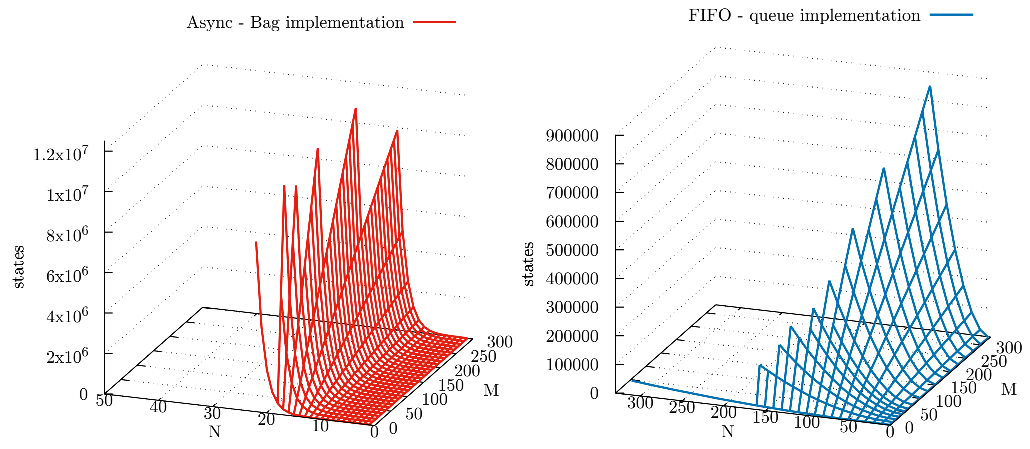


Figure 9: Number of Distinct States in the Studied System

to the number of repetitions of the scenario, thus the linear profile. These results show that in practice, systems should scale up well because high degrees of chaos, for instance when more than 20 messages are to transit on a communication medium at the same time ($N \gtrsim 20$), are seldom met.

5 Related Work

5.1 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 [ADF08]) 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 faulty reception completion (section 3.5.2), 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. The faulty reception completion ensures that fault transitions are introduced to get this property.

To sum up, although some works use 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 provide a tool to automatically check the composition. Compared to these works, we propose a unified formalization of several communication models and compatibility criteria, and a framework which makes it possible 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.

5.2 System Description

5.2.1 IO 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. IO automata can model asynchronous systems in a broad sense. IO automata provide a powerful framework to describe distributed systems, but are less practical to verify properties about them. Furthermore, few tools have been developed to make use of IO automata and perform modeling and property checking.

5.2.2 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 chose for its simplicity. Its main disadvantage for our work is that communications are synchronous, so we had to adapt its semantics. Milner also defined the π-calculus [Mil99]. The main difference is the introduction of parameters: channels can be communicated through channels themselves. This can describe systems with dynamic configurations. Still, the π-calculus is also synchronous.

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 can describe separated membranes/domains, where processes interact with each other within a domain or perform explicit actions to move into 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.

6 Conclusion and Perspectives

This paper presents a framework to check the compatibility of asynchronously communicating peers. It provides a general approach on the diversity of asynchronous communication and takes part in our current study and comparison of the asynchronous communication models. Their differences indeed play a major role in the compatibility of peer compositions as highlighted by the studied use case. The framework enables one to check concrete examples that hint at similarity between different models or reveal their differences.

In the considered approach, point-to-point asynchronous communication occurs between peers through channels without explicit sender and receiver. A communication model manages the communication events and induces properties on the transmission. Being able to associate channels of a peer composition to different communication models makes it possible to study which setup and which implementations, for given peers and compatibility properties, offer the lowest overhead or the fewest constraints. Formalizing and studying these notions is part of an ongoing work which also aims at automating the process. Extending the asynchronous models by introducing broadcast (analogous to a message consumed by more than one peer) and communication failures (mainly message loss) is planned too.

Finally, thanks to its modular conception and the reliance on transition systems, the framework is easily extensible and adaptable. Alternative ways to ease peer specification using CCS terms and a completion step on the derived transition system have been integrated to the automated toolchain. It accounts for the flexibility and adaptability of the developed tool. Benchmarking results also shows that the tool scales up well.

Bibliography

[ADF08] A. Ait-Bachir, M. Dumas, M.-C. Fauvet. BESERIAL: Behavioural Service Analyser. In *Business Process Management International Conference. Demo session.* Pp. 374–377. 2008. LNCS 5240.

[BB92] G. Berry, G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science* 96(1):217–248, 1992.

[BBO12] S. Basu, T. Bultan, M. Ouederni. Synchronizability for Verification of Asynchronously Communicating Systems. In *13th International Conference on Veri-*

fication, Model Checking, and Abstract Interpretation. VMCAI'12, pp. 56–71. Springer-Verlag, 2012.

[BCPV04] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo. Formalizing Web Service Choreographies. *Electronic Notes in Theoretical Computer Science* 105:73–94, Dec. 2004.

[BCT04] B. Benatallah, F. Casati, F. Toumani. Analysis and Management of Web Service Protocols. In *Conceptual Modeling – ER 2004*. Lecture Notes in Computer Science 3288, pp. 524–541. Springer, 2004.

[BZ83] D. Brand, P. Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM* 30(2):323–342, Apr. 1983.

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

[CL85] K. M. Chandy, L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems* 3(1):63–75, Feb. 1985.

[CLB08] H. S. Chae, J.-S. Lee, J. H. Bae. An Approach to Checking Behavioral Compatibility between Web Services. *International Journal of Software Engineering and Knowledge Engineering* 18(2):223–241, 2008.

[CMT96] B. Charron-Bost, F. Mattern, G. Tel. Synchronous, Asynchronous, and Causally Ordered Communication. *Distributed Compututing* 9(4):173–191, Feb. 1996.

[CPS93] R. Cleaveland, J. Parrow, B. 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, Jan. 1993.

[CPT01] C. Canal, E. Pimentel, J. M. Troya. Compatibility and inheritance in software architectures. *Science of Computer Programming* 41(2):105–138, Oct. 2001.

[DOS12] F. Durán, M. Ouederni, G. Salaün. A Generic Framework for N-protocol Compatibility Checking. *Science of Computer Programming* 77(7-8):870–886, July 2012.

[DWZ$^+$06] S. Deng, Z. Wu, M. Zhou, Y. Li, J. Wu. Modeling Service Compatibility with Pi-calculus for Choreography. In *25th International Conference on Conceptual Modeling*. Conceptual Modeling - ER 2006, pp. 26–39. Springer-Verlag, 2006.

[FG96] C. Fournet, G. Gonthier. The Reflexive CHAM and the Join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*. POPL '96, pp. 372–385. 1996.

[FGL$^+$96] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, D. Rémy. A Calculus of Mobile Agents. In Montanari and Sassone (eds.), *CONCUR*. Lecture Notes in Computer Science 1119, pp. 406–421. 1996.

[FUMK04] H. Foster, S. Uchitel, J. Magee, J. Kramer. Compatibility Verification for Web Service Choreography. In *IEEE International Conference on Web Services*. Pp. 738–. 2004.

[GGH$^+$10] P. Gardiner, M. Goldsmith, J. Hulance, D. Jackson, B. Roscoe, B. Scattergood, P. Armstrong. FDR2 User Manual. Technical report, Oxford University, Nov. 2010.

[Lam78] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM* 21(7):558–565, July 1978.

[Lam03] L. Lamport. *Specifying Systems*. Addison Wesley, 2003.

[LFS$^+$11] X. Li, Y. Fan, Q. Z. Sheng, Z. Maamar, H. 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] N. Lohmann, K. Wolf. Decidability Results for Choreography Realization. In *9th International Conference on Service-Oriented Computing*. ICSOC'11, pp. 92–107. Springer-Verlag, 2011.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Mar03] A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, pp. 12–20, 2003.

[Mil82] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Mil99] R. Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[Mis83] J. Misra. Detecting Termination of Distributed Computations Using Markers. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*. PODC '83, pp. 290–294. ACM, 1983.

[OSB13] M. Ouederni, G. Salaün, T. Bultan. Compatibility Checking for Asynchronously Communicating Software. In *International Symposium on Formal Aspects of Component Software (FACS 2013)*. LNCS 8348, pp. 310–328. 2013.

[RST91] M. Raynal, A. Schiper, S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters* 39:343–350, Oct. 1991.

[SM94] R. Schwarz, F. Mattern. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. *Distributed Computing* 7(3):149–174, June 1994.

[TFZ09] W. Tan, Y. Fan, M. 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.