A Framework for the Verification of Asynchronously Communicating Services

Florent Chevrou, Aurélie Hurault, Philippe Mauran, Philippe Quéinnec, and Xavier Thirioux

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

Abstract. Verifying the compatibility of services is a crucial issue in service oriented computing as this is a step to verify the correctness of the whole composition in which these services participate. In this paper, we present a framework to check services compatibility in the asynchronous world. We propose an approach that takes into account in a generic and unified way several compatibility notions and more importantly several asynchronous communication models (e.g. FIFO or causal). A system is composed of a set of services, which communicate via channels. A channel is not restricted to have a unique sender and a unique receiver. Moreover, channels can be partitioned into groups associated to different communication models, and thus different ordering properties, which constitutes a composite communication model. The notions of system, service, compatibility criteria and communication model are formalized in the TLA+ framework in order to benefit from its verification tools. As a result, a tool has been developed to generate the TLA+ specification from the services behavioral descriptions and to verify whether they are compatible for a given composite communication model and a given compatibility criterion.

Keywords: Service composition, formal verification, asynchronous communication, compatibility, TLA+

1 Introduction

Building systems through selecting, and then assembling and coordinating off-the-shelf components or services is a thriving 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 services is crucial to this approach. We consider this issue in the particular perspective of the development of distributed software systems. In this setting, the availability of the elementary services, as well as their interaction models (e.g. synchronous or asynchronous, multicast or point to point) can directly impact the properties of the global system, and especially its liveness properties. 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 [24], of compatibility of communicating components [5, 20]), 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. 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 algorithm correctness. For instance, Chandy-Lamport snapshot algorithm [10] requires that the communication between two processes is FIFO, and Misra termination detection algorithm [27] 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.

The presented work studies the effect of communication modalities on the properties of composite systems, in order to integrate the heterogeneity, the variability, and the diversity of interactions in distributed systems. This paper presents and illustrates the formal and methodological framework used to carry out this study. The outline of this paper is the following. Section 2 gives an intuition on the objectives and the method. Section 3 presents and formalizes the main notions: service, point to point communication model, and system. It also presents several classical communication models (FIFO, causal, ...). Section 4 shows how services are specified using CCS terms and used to generate the associated TLA+ modules [18] to which model checking is applied. It then defines the role of the compatibility properties in our framework with a model checking approach. Section 5 illustrates our approach with a simple case study, and provides the results obtained with TLC, the TLA+ model checker. Section 6 provides an overview of the conceptual background of this work and, eventually, the conclusion draws perspectives after summing up this work.

2 Intuition

Consider two services (or peers, or processes) 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 reception 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, 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 can say that the two services 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 service must be isolated from the other services: it does not have to be ready for all kind of messages. For instance, if the previous system also comprises two other services $\xrightarrow{c?}$ and $\xrightarrow{c!}$, a message on c may be in transit. However the communication medium will never deliver this message to the service $\xrightarrow{a?} \cdot \xrightarrow{b?}$, as this message does not concern it. In a given state, the interface of a service is defined as the set of messages it may consume later and thus are of interest. Only messages in its interface are delivered to a service.

One last point is that the services communicate through channels, and messages do not have an explicit destination process. One strong point is that we do not impose that channels have a unique sender and a unique receiver. Several services may send messages on the same channel, and several services may consume messages from the same channel. This allows to naturally describe arbitrary client-server and publish-subscribe architectures. For instance, several servers may consume from the same channel, allowing for distribution.

The goal of our framework is to verify various compatibility and incompatibility properties, such as the termination of all services (full compatibility), the possibility that an unexpected message is delivered to a service (leading to a fault), a forever blocking communication (a reception which never succeeds, or an emission which is continuously refused by the communication medium)... Specific properties of the communication media are also of interest: if the services terminate, is there any pending (unconsumed) message? Is the number of in transit messages bounded by a given value?

3 Formalization

3.1 Notation

In this paper, we mainly use the classic mathematical notation, and in a few cases, specific TLA+ notation [18] :

- Sequences and tuples are written $\langle a_1, a_2, a_3 \rangle$. $\langle\rangle$ is the empty sequence.
- In a transition predicate, 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".

3.2 System Model

Definition 1 (Service). *Let $\mathcal{C}$ be an enumerable set of channels. A service $\mathcal{S}_i$ is a labeled transition system $TS_i = (S_i, I_i, R_i, L_i)$ where S_i is the set of states, I_i is the set of initial states, L_i is a (enumerable) set of labels, and $R_i \subseteq S_i \times L_i \times S_i$ is the transition relation.*

The set of labels L_i contains τ and a subset of $\bigcup_{c \in \mathcal{C}} \{c!, c?\}$. τ is the usual internal action and we assume stuttering: $\forall s \in S_i : s \xrightarrow{\tau} s \in R_i$. The labels $c!$ and $c?$ will be 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 $\mathcal{S}_i$ (this means that $s \xrightarrow{c_1?} \_ \in R_i$ and $s \xrightarrow{c_2?} \_ \in R_i$), 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 service. 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.

Definition 2 (Listened channels). *Let s be a state in S_i,*

$$LC_i(s) \triangleq \{c \in \mathcal{C} \mid \exists s_2 \in S_i : s \xrightarrow{c?} s_2 \in R_i\}$$

Definition 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 (sates, 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 service i on channel c), and a subset of $\mathbb{N} \times \bigcup_{c\in\mathcal{C}}\{c?\} \times \mathcal{P}(\mathcal{C})$ (receive events by service i 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 3.3.

Definition 4 (Composed System). *The composed system $System = (S, I, R)$ is the product of the $TS_i : i \in 1..N$ with a communication model $\mathcal{CM}$*

- $S \subseteq S_1 \times \ldots \times S_N \times S_{CM}$
- $I = I_1 \times \ldots \times I_N \times I_{CM}$
- $R = \left\{ s \to s' \;\middle|\; \begin{pmatrix} \textbf{\textit{Internal actions}} \\ s_{cm} \xrightarrow{\tau} s'_{cm} \in R_{CM} \\ \wedge\ \forall i \in 1..N : s_i \xrightarrow{\tau} s'_i \in R_i \end{pmatrix} \vee \begin{pmatrix} \textbf{\textit{Communication}} \\ \exists i \in 1..N : \exists c \in \mathcal{C} : \\ \text{send}(s, s', i, c) \\ \vee\ \text{receive}(s, s', i, c) \end{pmatrix} \right\}$

Thus, a system state s is a tuple $(s_1, \ldots, s_N, s_{cm})$. Given a system state s, we note s_i the projection $\pi_i(s)$ of s on $\mathcal{S}_i$, and s_{CM} the projection of s on $\mathcal{CM}$.

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

$$\text{send}(s, s', i, c) \triangleq \begin{cases} s_i \xrightarrow{c!} s'_i \in R_i \\ s_{cm} \xrightarrow{i,c!} s'_{cm} \in R_{CM} \\ s_k \xrightarrow{\tau} s'_k \in R_k, \forall k \neq i, cm \end{cases}$$

$$\text{receive}(s, s', i, c) \triangleq \begin{cases} s_i \xrightarrow{c?} s'_i \in R_i \\ s_{cm} \xrightarrow{i,c?,L} s'_{cm} \in R_{CM} \text{ where } L = LC_i(s_i) \\ s_k \xrightarrow{\tau} s'_k \in R_k, \forall k \neq i, cm \end{cases}$$

To avoid infinite stuttering, we assume a minimal progress property: the services and the communication model can infinitely stutter only if no other transition can be done.

3.3 Asynchronous Communication Models

We informally describe five asynchronous communication models. Causal communication is formally presented, and the formalization of the other communication models is in appendix A, as it is not required to understand the paper. What follows is the logical descriptions of the communication models, not their implementations which must nonetheless fit these descriptions. Of course, an actual implementation of the system would use exact 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, which holds in transit messages. Depending on the communication model, this variable may be a queue, a set, a bag, an array of queues...

M_{unique} Unique FIFO This realizes a global fifo order: sent messages are put in a unique FIFO queue. Messages are globally ordered, and must be consumed in their send order. This model is unrealistic but is often a first step to decouple send and reception events.

M_{inst} Instantaneous FIFO Each service is equipped with a unique FIFO input queue. The sender instantaneously adds a message to this queue, without blocking. The receiver can remove from this queue later. This model is used for instance in [2, 28] as an abstraction of asynchronous communication. This model is as costly to implement as the synchronous communication model and is more restrictive than the next asynchronous models, as the order of received messages on a service is exactly the time-absolute order of their emissions, even when these emissions are totally independent. This means that if $\mathcal{S}_i$ consumes m_1 (sent by $\mathcal{S}_j$) and later m_2 (sent by $\mathcal{S}_k$), $\mathcal{S}_i$ ***knows*** that the sending on $\mathcal{S}_j$ occurs before the sending on $\mathcal{S}_k$ in the global execution order, even if there is no causal link between the two emissions.

M_{causal} Causally Ordered Communication Messages are delivered in an order compatible with the causality of their emission [17]. More precisely, if message m_1 is causally sent before m_2 (which means that there is a causal path from m_1 send event to m_2 send event), then they must not be consumed in a reverse order (which means that if they are consumed by the same service, m_1 cannot be consumed after m_2).

The state of the communication model is $(net, H_1, \ldots, H_N)$, composed of:
- *net*, the set of in transit messages
- for each service $\mathcal{S}_i$: H_i, the current causal past or history, that is the set of messages on which the next emission is causally dependent.

- A “message” is a couple $\langle$ channel, causal past of the message$\rangle$.
- $I_{CM} = (\emptyset, \emptyset, \ldots, \emptyset)$ (Initially, all sets are empty).
- Send action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c!} s'_{cm} \triangleq \begin{cases} H'_i \;=\; H_i \cup \{\langle c, H_i\rangle\} \\ net' \;=\; net \cup \{\langle c, H_i\rangle\} \\ \forall k \neq i : H'_K = H_k \end{cases}$$

- Receive action on $\mathcal{S}_i$: The message is not causally posterior to another in transit message which may be consumed by this service:

$$s_{cm} \xrightarrow{i,c?,L} s'_{cm} \triangleq \begin{cases} \exists \langle c, h\rangle \in net : \\ \quad \neg\exists\langle c_2, h_2\rangle \in net : c_2 \in L \wedge \langle c_2, h_2\rangle \in h \\ \quad net' = net \setminus \{\langle c, h\rangle\} \\ \quad H'_i = H_i \cup h \cup \{\langle c, h\rangle\} \\ \quad \forall k \neq i : H'_k = H_k \end{cases}$$

M_{fifo} FIFO Communication Messages from a same service to a same receiver are delivered in their emission order. Messages from different services are independently delivered. More precisely, if a service sends a message m_1 and later a message m_2, and these two messages are consumed by a same service, then m_2 cannot be consumed before m_1.

M_{async} Fully Asynchronous Communication This is unordered point-to-point communication. Messages are arbitrarily delivered, without any ordering.

Bounded Models The previous definitions can easily be adapted to introduce bounds on the size of *net* or on its projections (e.g. bounds on the number of in transit messages for each channel).

3.4 Composite Communication Models

Channels can be partitioned according to their communication properties. Each partition has its own communication model, and the global communication model composed of the communication models associated to each class is called a composite communication model. For instance, if the system uses five channels $\{c_1, c_2, c_3, c_4, c_5\}$, we can state that $\{c_1, c_4, c_5\}$ are causally ordered channels, and $\{c_2, c_3\}$ are fifo ordered channels. This means that messages on c_1, c_4, c_5 are all causally linked and ordered, whereas messages on c_2 and c_3 are fifo ordered. Messages on c_1 and c_2, being in different partitions, do not impose any constraint on each other.

4 Framework

We present in this section a framework aimed at checking compatibility properties over a composition of a set of services and a communication model (possibly

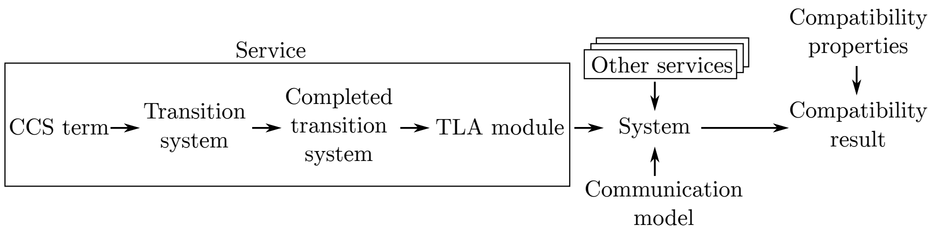

Fig. 1. Main Steps Performed by the Framework

composite). Services are specified using CCS terms [25] and the associated transition systems are computed. They are then completed to add implicit faulty receptions. The TLA+ specification consists of the conjunction of these transition systems and the communication model transition system.

Figure 1 provides an overview of the different steps and elements used to perform the verification of a service composition.

4.1 Services Specification

Definition 5 (Service Specification). *A service is a process specified by 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 an internal action* τ*, or a send action* $c!$ *over a channel* c*, or a receive action* $c?$ *on* c*,*
- *the choice operator* $+$*,*
- *the parallel composition operator* $\|$*,*
- *and process identifiers (defined by* $X \triangleq Process$*).*

We derive a service transition system (definition 1) from its specification using the standard CCS rules [26, p.39], excluding the synchronous communication rule. Renaming and restriction are currently not used but they would have no impact on the compatibility verification. Since we do not consider synchronous communication, $\|$ is similar to an interleaving operator. It can model internal parallelism and dynamic creation of processes inside a given service.

4.2 Faulty Reception Completion

The faulty reception completion (FRC) consists in revealing the unexpected receptions in a service and mark them as faulty by making the corresponding added transitions point toward a faulty state denoted $\perp$. The way transition systems are completed follows the intuition in section 2. Informally, for each state s where a reception transition exists, the interface of s, i.e. the set of channels corresponding to possible future receptions, is computed. For each channel c in the interface that is not already specified as an alternative choice in the current in-completion state, such a choice is provided by a transition toward $\bot$ and labeled by $c? : s \xrightarrow{c?} \bot$. These are called *faulty receptions*.

Definition 6 (Faulty Reception Completion). *Let* $TS = (S, I, R, L)$ *be a service.* $FRC(TS) \triangleq (S \cup \{\bot\}, I, R \cup R_2, L)$ *with*
$R_2 = \{s \xrightarrow{c?} \bot \mid s \in RS(S),\ c \in IC(s) \setminus LC(s)\}$*, and where*

- $\bot \notin S$ *(the added faulty state)*
- $RS(S) \triangleq \{s \in S \mid \exists c \in C, s' \in S : s \xrightarrow{c?} s' \in R\}$
  (states having at least one reception transition)
- $IC(s) \triangleq \{c \in C \mid \exists s_1, s_2 \in S : s \to s_1 \in R^* \wedge s_1 \xrightarrow{c?} s_2 \in R\}$
  (interface of s: future possible receptions)
- $LC(s) \triangleq \{c \in C \mid \exists s' \in S : s \xrightarrow{c?} s' \in R\}$ *(listened channels in state s)*

For instance, let us consider the initial state in the service specified by $a? \cdot b? \cdot 0$. The associated interface is $\{a, b\}$. Since in the initial state, there is no explicit alternative to handle $b?$, the state is completed with a $\xrightarrow{b?} \bot$ transition. The corresponding completed transition system is:

$$\begin{tikzcd} a? \cdot b? \cdot 0 \arrow[r, "a?"] \arrow[dr, "b?"'] & b? \cdot 0 \arrow[r, "b?"] & 0 \\ & \bot & \end{tikzcd}$$

When composed with a service $a! \cdot b! \cdot 0$ and an asynchronous communication model, both $a?$ and $b?$ first transitions are possible, and $b?$ leads to the faulty state $\bot$. When composed with a FIFO communication model, the first $b?$ transition is impossible (because a must be delivered before), and the system always reduces to 0.

4.3 Compatibility Checker

A compatibility property is given as an LTL formula over a system [22]. Let $System = (S, I, R)$ be a system. For a state $s = (s_1, \ldots, s_n, s_{cm}) \in S$, we define the following predicates:

- $0_\forall \triangleq \forall i \in 1..N : s_i = 0$ (all services are in their terminal state)
- $0_i \triangleq s_i = 0$ (termination of service i)
- $\bot_\exists \triangleq \exists i \in 1..N : s_i = \bot$ (an unexpected message has been delivered)

Inspired by section 2, the following compatibility properties are defined:

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

Service termination The service i always reaches a terminal state:
$System \models \Diamond\Box 0_i$

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

```
──────────────────────────── MODULE causal ────────────────────────────
EXTENDS Naturals
CONSTANTS CHANNEL, N
VARIABLES net, H
Init ≜ ∧ net = {} ∧ H = [i ∈ 1 .. N ↦ {}]
───────────────────────────────────────────────────────────────────────
TransitingMessages ≜ net ≠ {}
nochange ≜ UNCHANGED ⟨net, H⟩
internal ≜ FALSE

send(service, chan) ≜
      ∧ net′ = net ∪ {⟨chan, H[service]⟩}
      ∧ H′  = [H EXCEPT ![service] = @ ∪ {⟨chan, @⟩}]

receive(service, chan, listened) ≜ ∃ ⟨c1, H1⟩ ∈ net : (
     ∧ c1 = chan
     ∧ net′ = net \ {⟨chan, H1⟩}
     ∧ ¬(∃ ⟨c2, H2⟩ ∈ net : c2 ∈ listened ∧ ⟨c2, H2⟩ ∈ H1)
     ∧ H′ = [H EXCEPT ![service] = @ ∪ H1 ∪ {⟨chan, H1⟩}])
───────────────────────────────────────────────────────────────────────
```

Fig. 2. TLA+ Module Associated to the Causal Communication Model

No forever blocking communication No communication event is forever blocked (translated as no state is stable except termination and faulty reception):
$System \models \Box(0_\forall \vee \bot_\exists \vee \text{ENABLED}(R))$
where $\text{ENABLED}(R)$ is true iff a transition is possible in the current state.

4.4 TLA+ specifications

The TLA+ framework [18] allows to describe transition systems, to reason about them, and to verify LTL properties on them. A transition system is symbolically described with variables, and actions are used to describe the transition relation. TLA+ is based on the use of simple mathematics with the full expressive power of sets and functions.

In our framework, the TLA+ specification of a system consists of several modules:

Communication models They follow the specifications in 3.3. They consist of state variables representing the network state and, if necessary, histories. Send and receive actions are parameterized by the service identifier (used for instance by the FIFO communication model), the channel, and the listened channels in the case of reception. These actions follow the semantics of the models formal definition. Figure 2 shows the TLA+ module associated to the causal communication model as defined in 3.3.

Service management This module defines a vector data structure to represent and manipulate the services states using program counters. A service consists of a unique identifier and a program counters. Predicates are also specified to evaluate $0_\forall$ and $\bot_\exists$.

```
---------------------------- MODULE example ----------------------------
EXTENDS Naturals, servicemanagement
CONSTANTS N, b, a
VARIABLES net1, H1
Var1 ≜ ⟨net1, H1⟩
Vars ≜ ⟨peers, Var1⟩
Com ≜ INSTANCE fifo WITH CHANNEL ← {b, a}, N ← N
TypeInvariant ≜ Com!TypeInvariant ∧ ServiceTypeInvariant
-------------------------------------------------------------------------
Init ≜ ∧ Com!Init
       ∧ peers = ⟨11, 13⟩

t1(serv) ≜ trans(serv, 11, 12) ∧ Com!send(serv, a)                 state 11 $\xrightarrow{a!}$ state 12
t2(serv) ≜ trans(serv, 12, 0) ∧ Com!send(serv, b)                  state 12 $\xrightarrow{b!}$ 0
t3(serv) ≜ trans(serv, 13, 15) ∧ Com!receive(serv, a, {b, a})      state 13 $\xrightarrow{a?}$ state 15
t4(serv) ≜ trans(serv, 15, 0) ∧ Com!receive(serv, b, {b})          state 15 $\xrightarrow{b?}$ 0
t5(serv) ≜ trans(serv, 13, 1) ∧ Com!receive(serv, b, {b, a})       state 13 $\xrightarrow{b?}$ ⊥

Fairness ≜ ∀ i ∈ 1 .. N : (WF_Vars(t1(i)) ∧ ... ∧ WF_Vars(t5(i)))
             ∧ WF_Vars(Com!internal ∧ UNCHANGED peers)
Next ≜ ∃ i ∈ 1 .. N : (t1(i) ∨ ... ∨ t5(i)) ∨ (Com!internal ∧ UNCHANGED peers)
Spec ≜ Init ∧ □[Next]_Vars ∧ Fairness
=========================================================================
```

Fig. 3. $a! \cdot b! \cdot 0$ Composed with $a? \cdot b? \cdot 0$: Generated TLA+ Module

Composition For a set of services and a communication model, a TLA+ module is automatically generated according to the process described in 4.1. It provides transition actions by combining a transition from the communication model module and a transition from the service management module to update program counters accordingly. Figure 3 shows an example of such a TLA+ module. We can see that two services are initialized in states numbered 13 and 11. Two transitions departing from state 13 reveal that two alternatives are possible. One of them leads to state 1 after a reception from b, and corresponds to a faulty reception added during the FRC on $a? \cdot b? \cdot 0$.

5 Results

5.1 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 4 and the system is specified in figure 5.

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

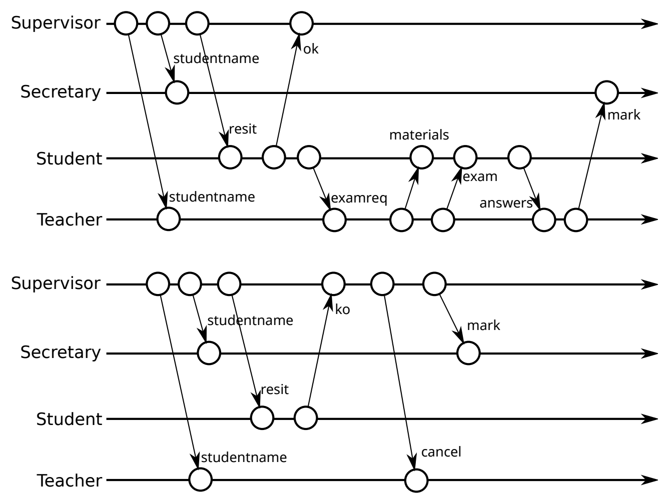

Fig. 4. Expected Executions Examples

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

Fig. 5. Supervisor-Secretary-Student-Teacher Specification

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.

We consider the five models defined in 3.3 and the composite model associated to the following partition:

	causal	$\{\mathit{studentname}, \mathit{resit}, \mathit{examreq}, \mathit{cancel}, \mathit{mark}\}$
	FIFO	$\{\mathit{materials}, \mathit{exam}\}$
(no constraint)	**async.**	$\{\mathit{ok}, \mathit{ko}, \mathit{answers}\}$

5.2 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

	Unique	Inst.	Causal	FIFO	Async.	Composite
Termination	✔	✔	✔	×	×	✔
Termination with an empty network	✔	✔	✔	×	×	✔
Partial termination (secretary)	✔	✔	✔	×	×	✔
No faulty receptions	✔	✔	✔	×	×	✔
No forever blocking communication	✔	✔	✔	×	×	✔

Fig. 6. Compatibility Results

be emitted by different services (supervisor and teacher). Therefore, compatibility, especially termination of the secretary service, is not trivial. Consequently, in addition to the already specified compatibility properties defined in 4.3, we also consider the termination of the secretary and we check if all messages have been received upon full termination.

Figure 6 presents the results. It shows that causality is needed to ensure compatibility of the composition. However it is not required over the whole set of channels. Indeed, the composite model with the considered partition is a restrictive enough communication model. In this example, with the chosen composite communication model, model checking generates 135 distinct states.

6 Related Work

6.1 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 [1]) or the notion of trace inclusion (as in [9]) is used to express this sameness. In this taxonomy, we can also include different models of failure traces [16], 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 [12, 9, 3, 13], process algebra [11, 6, 7], Petri nets [19, 29, 23]. Different criteria are used to represent compatibility: deadlock freedom [12, 13], unspecified receptions [5, 12], at least one execution leads to a terminal state [12, 3, 11, 19], all the executions lead to a terminal state [3, 6], no starvation [13], divergence [6]. Domain application conditions are also used [9, 7]. The communication models used are synchronous [12, 3, 13, 11, 6, 7] or instantaneous FIFO [2, 28].

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 several 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 can return a counterexample when a universal compatibility criteria is invalid.

6.2 System Description

IO automata Input/output automata [21] 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 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.

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 [25] 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 [26]. 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.

Richer process calculi exist, such as the Join-calculus [14] (and its extension to mobility [15]) based on the reflexive CHAM (CHemical Abstract Machine) [4] and also the Ambient calculus [8]. 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.

7 Conclusion

This paper presents a framework to check services compatibility. Its originality is to consider the asynchronous world, more complex but more realistic. Its key features are that a system can use different communication models (e.g. fifo or causal) for different groups of channels, and that channels are not restricted to have a unique sender and a unique receiver. Our framework is also parametric with regard to the compatibility criteria. The framework has been instantiated in TLA+ and thus benefits from its tools, especially TLC model checker. The TLA+ specifications are automatically generated from the service behavioral descriptions.

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

References

1. A. Ait-Bachir, M. Dumas, and M.-C. Fauvet. BESERIAL: Behavioural Service Analyser. In *Business Process Management International Conference. Demo session.*, pages 374–377, 2008. LNCS 5240.
2. S. Basu, T. Bultan, and M. 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.
3. B. Benatallah, F. Casati, and F. 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.
4. G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comput. Sci.*, 96(1):217–248, 1992.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, Apr. 1983.
6. A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. *Electron. Notes Theor. Comput. Sci.*, 105:73–94, Dec. 2004.
7. C. Canal, E. Pimentel, and J. M. Troya. Compatibility and inheritance in software architectures. *Sci. Comput. Program.*, 41(2):105–138, 2001.
8. L. Cardelli and A. D. Gordon. Mobile ambients. In *First International Conference on Foundations of Software Science and Computation Structure*, FoSSaCS '98, pages 140–155. Springer-Verlag, 1998.
9. H. S. Chae, J.-S. Lee, and 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.

10. K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, Feb. 1985.
11. S. Deng, Z. Wu, M. Zhou, Y. Li, and J. 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.
12. F. Durán, M. Ouederni, and G. Salaün. A generic framework for n-protocol compatibility checking. *Science of Computer Programming*, 77(7-8):870–886, July 2012.
13. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. In *IEEE International Conference on Web Services*, pages 738–, 2004.
14. C. Fournet and G. Gonthier. The reflexive cham and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*, POPL '96, pages 372–385. ACM, 1996.
15. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, 1996.
16. P. Gardiner, M. Goldsmith, J. Hulance, D. Jackson, B. Roscoe, B. Scattergood, and P. Armstrong. FDR2 user manual. Technical report, Oxford University, november 2010.
17. L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
18. L. Lamport. *Specifying Systems*. Addison Wesley, 2003.
19. X. Li, Y. Fan, Q. Z. Sheng, Z. Maamar, and H. Zhu. A Petri net approach to analyzing behavioral compatibility and similarity of web services. *Trans. Sys. Man Cyber. Part A*, 41(3):510–521, May 2011.
20. N. Lohmann and K. Wolf. Decidability results for choreography realization. In *9th International Conference on Service-Oriented Computing*, ICSOC'11, pages 92–107. Springer-Verlag, 2011.
21. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
22. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems – Specification*. Springer-Verlag, 1992.
23. A. Martens. On compatibility of web services. *Petri Net Newsletter*, pages 12–20, 2003.
24. B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, Oct. 1992.
25. R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
26. R. Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, New York, NY, USA, 1999.
27. J. 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.
28. M. Ouederni, G. Salaün, and T. Bultan. Compatibility checking for asynchronously communicating software. In *FACS'13*, volume 8348 of *LNCS*, pages 310–328, 2013.
29. W. Tan, Y. Fan, and M. Zhou. A Petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE T. Automation Science and Engineering*, 6(1):94–106, 2009.

A Formalization of the Asynchronous Communication Models

A.1 M_{unique} Unique FIFO

This realizes a global fifo order: all messages are put in a unique FIFO queue.

The state of the communication model is net, the queue of in transit messages.

- $I_{CM} = \langle\rangle$ (Initially, no message in transit)
- Send action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c!} s'_{cm} \triangleq net' = net \circ \langle c \rangle$$

- Receive action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c?,L} s'_{cm} \triangleq net = \langle c \rangle \circ net'$$

Observe that this looks like the fully asynchronous model (below), using a queue instead of a bag. However this one is the strongest communication model, whereas the fully asynchronous model is the weakest one.

A.2 M_{inst} Instantaneous FIFO

Each service is equipped with a unique FIFO input queue. The sender instantaneously adds a message to this queue, without blocking. The receiver can remove from this queue later. The order of received messages on a service is exactly the time-absolute order of their emissions, even when these emissions are totally independent.

The state of the communication model is (net, H) composed of:
- net, the set of in transit messages;
- H, the set of all sent messages (used as a history variable). Observe that this is a global variable, which is consistent with the fact that Instantaneous FIFO enforces a global ordering.

- A "message" is a couple $\langle$ channel, history of the message$\rangle$.
- $I_{CM} = (\emptyset, \emptyset)$ (Initially, all sets are empty).
- Send action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c!} s'_{cm} \triangleq \begin{cases} H' = H \cup \{\langle c, H \rangle\} \\ net' = net \cup \{\langle c, H \rangle\} \end{cases}$$

- Receive action on $\mathcal{S}_i$: The message is not time-absolute posterior to another in transit message.

$$s_{cm} \xrightarrow{i,c?,L} s'_{cm} \triangleq \begin{cases} \exists \langle c, h \rangle \in net : \\ \quad \neg\exists \langle c_2, h_2 \rangle \in net : c_2 \in L \wedge \langle c_2, h_2 \rangle \in h \\ \quad net' = net \setminus \{\langle c, h \rangle\} \\ \quad H' = H \end{cases}$$

A.3 M_{causal} Causally Ordered Communication

Described in the main part (section 3.3).

A.4 M_{fifo} FIFO Communication

The state of the communication model is $(net, H_1, \dots, H_N)$ composed of:

- net, the set of in transit messages;
- for each service $\mathcal{S}_i$: H_i, the current local emission past, that is the set of sent messages.

- A "message" is a triplet ⟨ channel, sender, history of the message⟩.
- $I_{CM} = (\emptyset, \emptyset, \dots, \emptyset)$ (Initially, all sets are empty)
- Send action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c!} s'_{cm} \triangleq \begin{cases} H'_i = H_i \cup \{\langle c, i, H_i\rangle\} \\ net' = net \cup \{\langle c, i, H_i\rangle\} \\ \forall k \neq i : H'_k = H_k \end{cases}$$

- Receive action on $\mathcal{S}_i$: The message is not posterior to another in transit message coming from the same service j.

$$s_{cm} \xrightarrow{i,c?,L} s'_{cm} \triangleq \begin{cases} \exists \langle c, j, h\rangle \in net : \\ \quad \neg\exists \langle c_2, l, h_2\rangle \in net : l = j \wedge c_2 \in L \wedge \langle c_2, l, h_2\rangle \in h \\ \quad net' = net \setminus \{\langle c, j, h\rangle\} \\ \quad \forall k : H'_k = H_k \end{cases}$$

A.5 M_{async} Fully Asynchronous Communication

The state of the communication model is net, the bag of in transit messages.

- $I_{CM} = \emptyset$ (Initially, no message in transit)
- Send action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c!} s'_{cm} \triangleq net' = net \cup \{\{\langle c\rangle\}\}$$

- Receive action on $\mathcal{S}_i$:

$$s_{cm} \xrightarrow{i,c?,L} s'_{cm} \triangleq \exists \langle c\rangle \in net : net' = net \setminus \{\{\langle c\rangle\}\}$$

Observe that this is exactly the same as the M_{unique} unique FIFO model, except that the net is a bag, instead of a fifo queue.