Towards correct Evolution of Conversation Protocols

Sarah Benyagoub
University of Mostaganem
benyagoub.sarah@univ-mosta.dz

Meriem Ouederni
IRIT-INP of Toulouse
meriem.ouederni@enseeiht.fr

Yamine Ait Ameur
IRIT-INP of Toulouse
yamine@enseeiht.fr

Distributed software systems change dynamically due to the evolution of their environment and/or requirements, their internal designing policies, and/or their specification bugs which must be fixed. Hence, checking system changes must be run continuously. Such systems are usually composed of distributed software entities (called peers) interacting with each other through message exchanges, and this is to fulfil a common goal. The goal is often specified by a conversation protocol (CP), *i.e.* **sequences of sent messages. If there exists a set of peers implementing CP, then CP is said to be realisable. In this paper, we propose a stepwise approach for checking whether an evolution,** *i.e.* **adding and/or removing messages and/or peers, can be applied to a CP that was realisable before updating it. We define a set of correct evolution patterns and we suggest an algebra of CP evolution. Our approach ensures that CP evolution preserves the realisability condition.**

System Evolution, Realisability, Conversation Protocols, Formal Verification, Behavioural Systems

1. INTRODUCTION

Distributed software systems change dynamically due to the evolution of their running environment and/or requirements, their internal designing policies, and/or their possible specification bugs which must be fixed. Such systems are usually composed of distributed software entities (called peers) evolving concurrently in a distributed setting and interacting with each other throughout messages exchanges to fulfil a common goal.

In a top-down design of distributed software, the interaction among peers is usually modelled using collaboration diagrams, Message Sequence Chains (MSCs) or conversation protocols (CP) (Bultan 2006). Let us focus on CPs, these describe interactions among peers by describing uniquely the allowed sequences of sent messages. Here, it is crucial to know if the set of interactions in a CP can be implemented. In other word, considering a CP, one must check whether there exists a set of peers where their composition generates the same sequences of send messages as specified by the CP. This issue characterises the realisability problem.

In order to formally specify, verify, and fix issues which violate realisability, CP can be modelled using Labelled Transition Systems (LTSs) where communication follows either synchronous or asynchronous semantics. Using this model enables automated analysis of interaction properties, *e.g.*, realisability checking. Although it is obvious to check realisability in the case of synchronous communication, this realisability problem remains undecidable in the most general setting (Brand and Zafiropulo 1983) due the possibly ever-increasing queuing mechanism and unbounded buffers in the case of asynchronous communication.

The recent work of (Basu et al. 2012) proposed a necessary and sufficient condition for verifying if a CP can be implemented by a set of peers communicating asynchronously throughout FIFO buffers with no restriction on their buffer sizes. This work solves the realisability issue for a subclass of asynchronously communicating peers, namely, the synchronisable systems, *i.e.*, the system composed of interacting peers behaves equally by applying synchronous or asynchronous communication. A CP is realisable if there exists a set of peers implementing that CP, *i.e.*, they send messages to each other in the same order as CP, and such that their composition is synchronisable. In (Basu et al. 2012), the full checking of CP realisability applies the following steps: i) peer projection from CP; ii) checking synchronisability; and iii) checking equivalence between CP and its distributed system.

Based on LTS model, we can verify CP realisability using existing techniques such as model checking for systems with reasonable sizes (Basu et al.

2012) (*i.e.*, number of states, transitions and communicating peers) or theorem proving for scalable systems (Farah et al. 2016).

Considering realisable CPs, we are interested in studying the evolution of those CPs. In fact, these specify cross-organisational interactions with no centralised control between peers which can be administrated and executed by geographically distributed and autonomous companies. In order to cope with new environment changes and end-user requirements, system interaction and the corresponding CP need to evolve continuously over time. However, changing CP might result in knock-on effects on its realisability. Hence, verifying the correctness of CP evolution to ensure realisability preservation must also be run continuously.

Regarding the literature, existing work such as (Rinderle et al. 2006b; Ryu et al. 2008; Roohi and Salaün 2011) give some solutions for system evolution. In (Rinderle et al. 2006b; Ryu et al. 2008) the authors propagate the choreography updates into communicating peers. (Roohi and Salaün 2011) focuses on system reconfiguration meaning that for a CP which has been updated into CP', the authors check whether the traces that has been executed in CP can be performed again in CP'. This reconfiguration can be better applied for run-time system to ensure execution consistency. All these approaches do not consider realisability preservation.

There exist other research approaches which can be applied as a posteriori evolution checking. The approaches suggest solutions every time the realisability check fails. For example, *existing work on enforcing CP realisability such as the one given in (Güdemann et al. 2012) and recently on CP repairability (Basu and Bultan 2016) can be used to ensure the realisability of an already updated CP.*

Our statement is different than existing work and it is as follows: an evolution is allowed if it does not violate the CP realisability. By doing so, we suggest a priori verification approach of CP evolution. Instead of running the full realisability checking as described previously and detailed in Section 2, our proposal consists in performing partial verification uniquely at the CP level in order to answer the question if there *still* exist a set of peers implementing the updated CP. In this work, we consider the evolution at the CP level and we study its realisability effect on the distributed peers. The main issue is considering that system specifications may change over time (*e.g.*, service upgrade or degrade by adding and/or removing either messages exchanges or interacting peers), how can we ensure realisability preservation? to answer these questions, we proceed as follows:

- We first describe CP using LTS
- We rely on the realisability condition given in (Basu et al. 2012)
- We identify the set of behavioural properties which can hold by CP evolution yet they violate the realisability condition
- We suggest a set of evolution patterns and we show how the application of such patterns do not violate CP realisability
- We propose a language for correct CP evolution

The remainder of this paper is structured as follows: Section 2 introduces the formal definitions and the background on which our proposal relies. Section 3 presents the behavioural properties to be checked before application of CP evolution. In Section 4, we suggest an algebra for CP evolution with no violation of realisability. We present in Section 5 a case study to illustrate our approach. Section 6 overviews related work. Lastly, we conclude our work and present some perspectives in Section 7.

2. DEFINITIONS

In this section, we present our behavioural model for peers and CP. We, then, define how distributed peers can be obtained by projection from a given CP. Lastly, we define synchronisable systems, and we present realisability condition considering asynchronous communication.

2.1. Peer Model

We use Labeled Transition Systems (LTSs) for modelling the CP and the peers included in that specification. This behavioural model defines the order of sent messages in CP. At the peers level, the LTS can be computed by projection from CP and they describe the order in which those peers execute their send and receive actions.

Definition 1 (Peer) *A peer is an LTS* $\mathcal{P} = (S, s^0, \Sigma, T)$ *where* S *is a finite set of states,* $s^0 \in S$ *is the initial state,* $\Sigma = \Sigma^! \cup \Sigma^? \cup \{\tau\}$ *is a finite alphabet partitioned into a set of send messages, receive messages, and the internal action, and* $T \subseteq S \times \Sigma \times S$ *is a transition relation.*

We write $m!$ for a send message $m \in \Sigma^!$ and $m?$ for a receive message $m \in \Sigma^?$. We use the symbol τ (tau in figures) for representing internal activities. A transition is represented as $s \xrightarrow{l} s'$ where $l \in \Sigma$.

Example 1 *The right side of Figure 1 shows an example of three peers LTSs.*

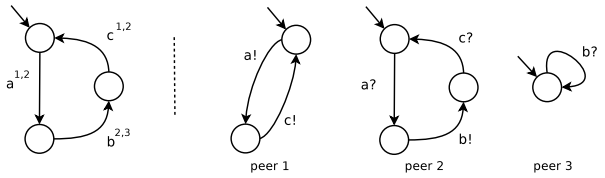

Figure 1: *CP (left side), Peers (right side)*

Definition 2 (Conversation Protocol : CP) *A conversation protocol CP for a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is an LTS $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ where S_{CP} is a finite set of states and $s^0_{CP} \in S_{CP}$ is the initial state; L_{CP} is a set of labels where a label $l \in L_{CP}$ is denoted $m^{\mathcal{P}_i,\mathcal{P}_j}$ such that $\mathcal{P}_i$ and $\mathcal{P}_j$ are the sending and receiving peers, respectively, $\mathcal{P}_i \neq \mathcal{P}_j$, and m is a message on which those peers interact; finally, $T_{CP} \subseteq S_{CP} \times L_{CP} \times S_{CP}$ is the transition relation. We require that each message has a unique sender and receiver: $\forall(\mathcal{P}_i, m, \mathcal{P}_j), (\mathcal{P}'_i, m', \mathcal{P}'_j) \in L_{CP} : m = m' \implies \mathcal{P}_i = \mathcal{P}'_i \wedge \mathcal{P}_j = \mathcal{P}'_j$.*

In the remainder of this paper, we denote a transition $t \in T_{CP}$ as $s \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s'$ where s and s' are source and target states and m^{P_i,P_j} is the transition label.

Example 2 *The left side of Figure 1 shows an example of CP LTS.*

Definition 3 (Projection) *Peer LTSs $\mathcal{P}_i =< S_i, s^0_i, \Sigma_i, T_i >$ are obtained by replacing in $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$ each label $(\mathcal{P}_j, m, \mathcal{P}_k) \in L_{CP}$ with $m!$ if $j = i$ with $m?$ if $k = i$ and with τ (internal action) otherwise; and finally removing the τ-transitions by applying standard minimisation algorithms Hopcroft and Ullman (1979).*

Example 3 *Notice that the peers on Figure 1 are obtained by projection from the CP shown on left side of the same Figure.*

2.2. Synchronous Composition

The synchronous composition of a set of peers corresponds to the system in which the peer LTSs communicate using synchronous communication. In this context, a communication between two peers occurs if both agree on a synchronisation label, *i.e.*, if one peer is in a state in which a message can be sent, then the other peer must be in a state in which that message can be received. One peer can evolve independently from the others through an internal action.

Definition 4 (Synchronous System) *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ with $\mathcal{P}_i = (S_i, s^0_i, \Sigma_i, T_i)$, the synchronous system $(\mathcal{P}_1 \mid \ldots \mid \mathcal{P}_n)$ is the LTS (S, s^0, Σ, T) where:*

- $S = S_1 \times \ldots \times S_n$
- $s^0 \in S$ *such that* $s^0 = (s^0_1, \ldots, s^0_n)$
- $\Sigma = \cup_i \Sigma_i$
- $T \subseteq S \times \Sigma \times S$, *and for* $s = (s_1, \ldots, s_n) \in S$ *and* $s' = (s'_1, \ldots, s'_n) \in S$

(interact) $s \xrightarrow{m} s' \in T$ *if* $\exists i, j \in \{1, \ldots, n\} : m \in \Sigma^!_i \cap \Sigma^?_j$ *where* $\exists\ s_i \xrightarrow{m!} s'_i \in T_i$, *and* $s_j \xrightarrow{m?} s'_j \in T_j$ *such that* $\forall k \in \{1, \ldots, n\}, k \neq i \wedge k \neq j \Rightarrow s'_k = s_k$

2.3. Asynchronous Composition

In the asynchronous composition, the peers communicate with each other asynchronously through FIFO buffers. Each peer $\mathcal{P}_i$ is equipped with an unbounded message buffer Q_i. A peer can either send a message $m \in \Sigma^!$ to the tail of the receiver buffer Q_j at any state where this send message is available, read a message $m \in \Sigma^?$ from its buffer Q_i if the message is available at the buffer head, or evolve independently through an internal action. Since reading from the buffer is not considered as an observable action, it is encoded as an internal action in the asynchronous system.

Definition 5 (Asynchronous Composition) *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ with $\mathcal{P}_i = (S_i, s^0_i, \Sigma_i, T_i)$,*

and Q_i being its associated buffer, the asynchronous composition $((\mathcal{P}_1, Q_1) \parallel \ldots \parallel (\mathcal{P}_n, Q_n))$ is the labeled transition system $LTS_a = (S_a, s^0_a, \Sigma_a, T_a)$ where:

1. $S_a \subseteq S_1 \times Q_1 \times \ldots \times S_n \times Q_n$ *where* $\forall i \in \{1, \ldots, n\},\ Q_i \subseteq (\Sigma^?_i)*$
2. $s^0_a \in S_a$ *such that* $s^0_a = (s^0_1, \epsilon, \ldots, s^0_n, \epsilon)$ *(where ϵ denotes an empty buffer)*
3. $\Sigma_a = \cup_i \Sigma_i$
4. $T_a \subseteq S_a \times \Sigma_a \times S_a$, *and for* $s = (s_1, Q_1, \ldots, s_n, Q_n) \in S_a$ *and* $s' = (s'_1, Q'_1, \ldots s'_n, Q'_n) \in S_a$

   (send) $s \xrightarrow{m!} s' \in T_a$ *if* $\exists i, j \in \{1, \ldots, n\}\ where\ i \neq j : m \in \Sigma^!_i \cap \Sigma^?_j$, *(i)* $s_i \xrightarrow{m!} s'_i \in T_i$, *(ii)* $Q'_j = Q_j m$, *(iii)* $\forall k \in \{1, \ldots, n\} : k \neq j \Rightarrow Q'_k = Q_k$, *and (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

   (consume) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1, \ldots, n\} : m \in \Sigma^?_i$, *(i)* $s_i \xrightarrow{m?} s'_i \in T_i$, *(ii)* $mQ'_i = Q_i$, *(iii)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow Q'_k = Q_k$, *and (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(internal) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1,\ldots,n\}$, *(i)* $s_i \xrightarrow{\tau} s'_i \in T_i$, *(ii)* $\forall k \in \{1,\ldots,n\} : Q'_k = Q_k$, *and (iii)* $\forall k \in \{1,\ldots,n\} : k \neq i \Rightarrow s'_k = s_k$

We use LTS_a^k to define the *bounded asynchronous composition*, where each message buffer is bounded to size k. The definition of LTS_a^k can be obtained from Def. 5 such that the maximum number of send messages that can be stored in the buffers is limited to k. A system is synchronizable (Basu et al. 2012) when its behavior remains the same under both synchronous and asynchronous communication semantics. This is checked by bounding buffers to $k = 1$ and comparing interactions in the synchronous system with send messages in the asynchronous system.

Definition 6 (Synchronizability) *Given a set of peers* $\{\mathcal{P}_1,\ldots,\mathcal{P}_n\}$, *the synchronous system* $(\mathcal{P}_1 \mid \ldots \mid \mathcal{P}_n) = (S_s, s_s^0, L_s, T_s)$, *and the 1-bounded asynchronous system* $((\mathcal{P}_1, Q_1^1) \mid\mid \ldots \mid\mid (\mathcal{P}_n, Q_n^1)) = (S_a, s_a^0, L_a, T_a)$, *two states* $r \in S_s$ *and* $s \in S_a$ *are synchronizable if there exists a relation* $Sync$ *such that* $Sync(r,s)$ *and:*

- *for each* $r \xrightarrow{m} r' \in T_s$, *there exists* $s \xrightarrow{m!} s' \in T_a$, *such that* $Sync(r',s')$;
- *for each* $s \xrightarrow{m!} s' \in T_a$, *there exists* $r \xrightarrow{m} r' \in T_s$, *such that* $Sync(r',s')$;
- *for each* $s \xrightarrow{m?} s' \in T_a$, $Sync(r,s')$.

The set of peers is synchronizable if $Sync(s_s^0, s_a^0)$.

Example 4 *The system described in Figure 1 is not synchronisable because peer 1 can send "a" and "c" in sequence in the asynchronous system, whereas "b" occurs before "c" in the synchronous system as specified in the CP.*

In order to check CP realisability, there is a need to check well-formedness. Well-formedness states that whenever the i-th peer buffer Q_i is non-empty, the system can eventually move to a state where Q_i is empty. For every synchronizable set of peers, if the peers are deterministic, *i.e.*, for every state, the possible send messages are unique, well-formedness is implied.

The approach presented in Basu et al. (2012) proposes a sufficient and necessary condition showing that the realizability of conversation protocols is decidable.

Definition 7 (Realizability) *A conversation protocol* CP *is realizable if and only if (i) the peers computed by projection from this protocol are synchronizable, (ii) the 1-bounded system resulting from the peer composition is well-formed, and (iii) the synchronous version of the distributed system* $\{\mathcal{P}_1,\ldots,\mathcal{P}_n\}$ *is equivalent to* CP.

In the remainder of this paper, we refer to a realisable CP as R(CP).

Both synchronizability and realizability properties can be checked automatically using equivalence checking as done in Basu et al. (2012). This check requires the modification of the asynchronous system for hiding receptions ($m? \rightsquigarrow \tau$), renaming emissions into interactions ($m! \rightsquigarrow m$), and removing τ-transitions using standard minimization techniques. The correctness of that approach is given in (Farah et al. 2016).

3. BEHAVIOURAL PROPERTIES

In order to check the realisability of a CP that has been updated, we must ensure that the resulting LTS does not hold some branching and/or sequential structures which violate realisability condition. We define in the following some properties which enable us to check such structures.

3.1. Branches related Properties

Property 1 (Non-Deterministic Choice (NDC)) *Given a conversation protocol* $CP =< S_{CP}, s_{CP}^0, L_{CP}, T_{CP} >$, *a state* $s_{CP} \in S_{CP}$ *is called non-deterministic branching state if :*

- $\exists\{s_{CP} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s'_{CP},\ s_{CP} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s''_{CP}\} \subseteq T_{CP}$, *and*
- $s'_{CP} \neq s''_{CP}$

This choice is referred to as non-deterministic choice.

We define in the following divergent choice (this is also called non-local branching choice in the literature) and it is different than process divergence (Ben-Abdallah and Leue 1997).

Property 2 (Divergent-Choice) *Given a conversation protocol* $CP =< S_{CP}, s_{CP}^0, L_{CP}, T_{CP} >$, *a state* $s_{CP} \in S_{CP}$ *is divergent branching state if :*

- $\exists\{s \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s'_{CP},\ s_{CP} \xrightarrow{m'^{\mathcal{P}_j,\mathcal{P}_i}} s''_{CP}\} \subseteq T_{CP}$, *and*
- $s'_{CP} \neq s''_{CP}$, *and*
- $m \neq m'$

This choice is referred to as divergent choice.

3.2. Sequences related Properties

Given a CP, there is at least two partitions of peers where there exist no interaction between both partitions.

Property 3 (Independent Sequences (ISeq)) *Given a conversation protocol* $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$*, a transition sequence* $s_{CP} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} \dots s'_{CP} \xrightarrow{m'^{\mathcal{P}_k,\mathcal{P}_q}} s''_{CP}$*, where all transitions are in* T_{CP}*, is called independent sequence if:*

- $\{P_i, P_j\} \cap \{P_k, P_q\} = \varnothing$

The following property enables us to detect sequences in CP which lead to non-local emission choices made by two different peers in the distributed system. To avoid that situation, every peer that join the conversation at an intermediate state (*i.e.*, different than the initial state) must be receiver the first time it shows up. Otherwise, if a peer would like to send a message m at an intermediate state, then this must be receiver in its last interaction before sending m.

Property 4 (Divergent Sequences (DSeq)) *Given a CP, there exists a transition sequence* $s^0_{CP} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} \dots s'_{CP} \xrightarrow{m'^{\mathcal{P}_k,\mathcal{P}_q}} s''_{CP}$ *where all transitions are in* T_{CP}*:*

- *for every sender peer* P_t *appearing before state* s'_{CP}*,* $t \neq k$*, or*
- *there is at least a transition* $s_{CP} \xrightarrow{m^{\mathcal{P}_k,\mathcal{P}_t}} s'''_{CP} \in T_{CP}$ *such that:*
  - s'_{CP} *is reachable from* s_{CP}*, and*
  - *there is no transition in* $s_{CP} \xrightarrow{m^{\mathcal{P}_k,\mathcal{P}_t}} s'''_{CP} \dots s' \xrightarrow{m'^{\mathcal{P}_k,\mathcal{P}_q}} s''_{CP}$ *where* P_k *is receiver.*

4. COMPOSITIONAL REALISABILITY

CP evolution stands for two possible tasks, namely, adding and/or removing either messages and/or interacting peers. We define here how CP realisability can be preserved by applying some evolution patterns presented in the following.

4.1. Evolution Patterns

We introduce in this paper two composition operators denoted as $\otimes_{(+,s_{CP})}$ for branching composition and $\otimes_{(\gg,s_{CP})}$ for sequential composition. We also assume other operators not presented here for lack of space, namely, $\otimes_{(\|,s_{CP})}$ for parallel composition, and $\otimes_{(\circlearrowleft,s_{CP})}$ for looping composition. The operator $\otimes_{(\|,s_{CP})}$ generates at a state s_{CP} all the interleaved behaviour of a set of transitions such that every generated branch must satisfy sequence related properties. The operator $\otimes_{(\circlearrowleft,s_{CP})}$ enables us to add self-loop of the form $s \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s$ where $i \neq j$ and such that sequence related properties must be preserved.

Definition 8 $\otimes_{(\gg,s_{CP})}$ *Given a* $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$*, a* $CP' =< S_{CP'}, s^0_{CP'}, L_{CP'}, T_{CP'} >$ *and a state* $s_{CP} \in S_{CP}$*, the sequential composition* $\otimes_{(\gg,s_{CP})}(CP, CP')$ *means that* CP *must be executed before* CP' *such that Properties 3 and 4 do not hold.*

Definition 9 $\otimes_{(+,s_{CP})}$ *Given a* $CP =< S_{CP}, s^0_{CP}, L_{CP}, T_{CP} >$*, a set* $\{CP'_i\}$*,* $i = 1..n$ *such that* $CP'_i =< S_{CP'_i}, s^0_{CP'_i}, L_{CP'_i}, T_{CP'_i} >$ *and a state* $s_{CP} \in S_{CP}$*, the branching composition* $\otimes_{(+,s_{CP})}(CP, \{CP'_1, \dots, CP'_n\})$ *means that there is a choice at* s_{CP} *between the remaining behaviour of CP (i.e., starting from* s_{CP}*) and all* CP'_i *such that:*

- *Properties 1 and 2 do not hold at the state* s_{CP}*, and*
- $\forall CP'_i,\ \otimes_{(\gg,s_{CP})}(CP, CP'_i)$ *holds*

Remark 1 *The application of an operator* $\otimes_{(op,s_{CP})}(CP, CP')$ *assumes that the initial state of* CP' *is fused with the state* s_{CP}*.*

4.2. Algebra for CP Evolution: syntax and language

We introduce in Listing 1 a CP algebra which we use for defining the evolution such that realisability is preserved. We refer to a state s^f as final if there is no outgoing transition at that state. We denote by ECP a CP that evolutes over time while preserving realisability. The expression ECP^+ stands for one or more ECP.

$$ECP ::= ECP_b \mid ECP\ op\ {ECP_b}^+$$

$$ECP_b ::= s \xrightarrow{(\mathcal{P}_i,m,\mathcal{P}_j)} s' \mid \varnothing$$

$$op ::= \otimes_{(+,s^f)} \mid \otimes_{(\gg,s^f)} \mid \otimes_{(\|,s^f)} \mid \otimes_{(\circlearrowleft,s^f)}$$

Listing 1: CP Evolution Grammar

4.3. Realisable CP Evolution

Conjecture 1 ECP_b *is realisable.*

Proof 1 *This is obvious by default.*

Conjecture 2 *Given an* $ECP =< S_{ECP}, s^0_{ECP}, L_{ECP}, T_{ECP} >$ *and a* ECP_b *such that* $R(ECP)$ *and* $R(ECP_b)$*,* $s^f \in S_{ECP}$*, then*

$$R(\otimes_{(\gg,s^f)}(ECP, ECP_b)) \quad (1)$$

Sketch 1 *This follows from Definition 7 and Properties 3 and 4. .* ■

Conjecture 3 *Given an* $ECP =< S_{ECP}, s^0_{ECP}, L_{ECP}, T_{ECP} >$ *and a set of* $n \in \mathbb{N}$ ECP_{bi} $(i \in [0..n-1])$ *such that* $R(ECP)$ *and* $R(ECP_b)$, $s^f \in S_{ECP}$, *then*

$$R(\otimes_{(+,s^f)}(ECP, \{ECP_{b0}, \ldots, ECP_{bn-1}\})) \quad (2)$$

Sketch 2 *This follows from Definition 7 and Properties 1 and 2.* ■

We denote by $\widehat{ECP}$ the set of all conversation protocols resulting from any evolution and such that the realisability is preserved. $\widehat{ECP}$ can be obtained by inductive composition using the aforementioned operators and the grammar.

Definition 10 (Correct Evolution) *Given a CP and* n CP_b *where* $n \in \mathbb{N}$ *,* CP_{bi} *stands for the* i^{th} CP_b *such that* $i \in [0..n-1]$, *and* $R(CP)$, *then:*

$$\widehat{ECP} = \{ECP \mid \forall op_j, j \in [0..n], op_j \in \textbf{OP},$$
$$ECP = op_0 \ \ldots \ op_n(CP \ CP_{b0} \ldots \ CP_{bn-1})$$
$$\}$$

In the evolution context, an initial CP can be updated into an ECP if and only if $ECP \in \widehat{ECP}$.

We generalise in the following the result given previously in this section.

Conjecture 4

$$\frac{R(CP_b), op \in \textbf{OP} \ \frac{R(CP)}{R(op(CP, CP_b))}}{\forall ECP \in \widehat{ECP}, \ R(ECP)} \quad (3)$$

Sketch 3 *It follows from Conjectures 1, 2, and 3 and Definition 10.* ■

5. CASE STUDY

5.1. Toy Example

This section shows some examples to better understand our proposal. We first give an illustration of CP evolution using a toy example where an initial CP and a possible evolution of that CP are shown on Figures 2 and 3, respectively. Notice that the added behaviour is presented with dashed transitions on Figure 3.

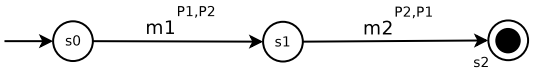

Figure 2: *Toy CP*

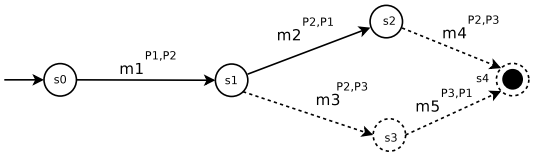

Figure 3: *Toy ECP*

Valid Evolution. In this example, the evolution is valid and can be applied since it is possible to generate ECP by application of the set of production rules from the evolution grammar (Listing 1) as follows. we try to rewrite ECP starting from the initial state of CP:

$$CP = s0 \xrightarrow{m_1^{P1,P2}} s1$$

$$CP_{b0} = s1 \xrightarrow{m_2^{P2,P1}} s2$$

$$CP_{b1} = s1 \xrightarrow{m_3^{P2,P3}} s3$$

$$CP_{b2} = s2 \xrightarrow{m_4^{P2,P3}} s4$$

$$CP_{b3} = s3 \xrightarrow{m_5^{P3,P1}} s4$$

Production Rules for toy ECP

$ECP = \otimes_{(\gg,s3)}($
$\otimes_{(\gg,s2)}($
$\otimes_{(+,s1)}(CP, \{CP_{b0}, CP_{b1}\})$
$, CP_{b2})$
$CP_{b3})$

5.2. Real Example

For illustration purposes we specify the use of an application in the cloud. This system involves four peers: a client (cl), a Web interface (int), a software application (appli), and a database (db). We show first a conversation protocol (Figure 4) describing the requirements that the designer expects from the composition-to-be. The conversation protocol starts with a login interaction (connect) between the client and the interface, followed by the access request (access) triggered by the client. This request can

be repeated as far as necessary. Finally, the client decides to logout from the interface (logout)

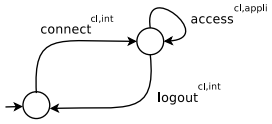

Figure 4: *A realisable CP*

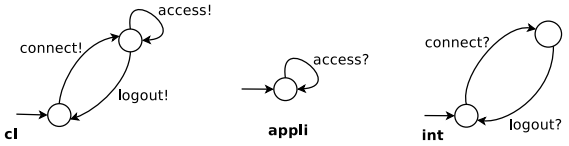

Figure 5: *Peers Projection from CP in Figure 4*

Invalid Evolution. We show on Figure 6 an updated version of the CP given on Figure 4 describing the new requirements that the designer expects from the composition-to-be. The conversation protocol starts with a login interaction (connect) between the client and the interface, followed by the setup of the application triggered by the interface (setup). Then, the client can access and use the application as far as necessary (access). Finally, the client decides to logout from the interface (logout) and the application stores some information (start/end time, used resources, etc.) into a database (log).

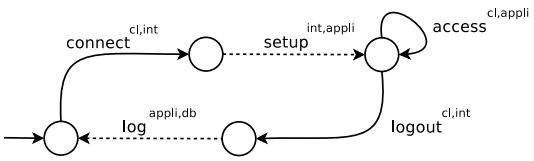

Figure 6: *A non correct CP Evolution*

Figure 7 shows the four peers obtained by projection. This set of peers seems to respect the behaviour specified in the conversation protocol, yet this is difficult to be sure using only visual analysis, even for such a simple example. In addition, as the CP involves looping behaviour, it is hard to know whether the resulting distributed system is bounded and finite, which would allow its formal analysis using existing verification techniques. Actually, this set of peers is not synchronisable (and therefore not realisable), because the trace of send messages "connect, access" is present in the 1-bounded asynchronous system, but is not present in the synchronous system. Synchronous communication enforces the sequence "connect, setup, access" as specified in the CP, whereas in the asynchronous system peer cl can send connect! and access! in sequence.

This kind of evolution resulting in non realisable CP can be avoided using our method with no need of CP

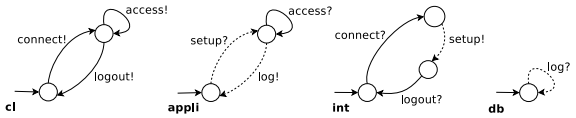

Figure 7: *Peers Projection from CP on Figure 4*

projection. Starting from the initial state of CP that is shown on Figure 4 and using the grammar given in Listing 1, there is no way to generate the interaction sequence "connect, setup, access" because adding "access" interaction violates Property 4.

6. RELATED WORK

Studying the evolution of software systems is not a new topic in itself. For instance, as stated in (Roohi and Salaün 2011), dynamic reconfiguration (Medvidovic 1996) is dealt with in the context of distributed systems and software architectures, graph transformation, software adaptation, or metamodelling. However, these do not study systems where their interaction is described with conversation protocols. In this section we restrict our study to the related approaches focusing on the evolution of conversation protocols.

(Leite et al. 2013) surveys the approaches working on Web service evolution until 2013. Our study of that work shows that the most related proposals are (Roohi and Salaün 2011; Wombacher 2009; Jureta et al. 2007; Rinderle et al. 2006a,b).

(Roohi and Salaün 2011) suggest a method to check CP reconfigurability. The authors consider two CPs, namely, an initial CP and a new CP' and two sets of peers PS and PS' obtained by projection from both CPs, respectively. Given a trace t in CP which consists in the history of the current execution (*i.e.*, the sequence of interactions held between the peers), if t can also be executed in reconfigured peers generated from CP', then the reconfiguration can take place.

Wombacher (2009) uses a formal model based on annotated Finite State Automata (aFSA) to describe Web service interaction and which is specified using choreography. This approach aligns changes between updated choreography and the correspondent orchestration. Given an updated choreography, the changes, namely, adding and/or removing sequences of messages are propagated into distributed peers. The proposed solution is implemented into DYCHOR framework and needs human validation. In (Rinderle et al. 2006a,b), the authors propose a controlled evolution method where propagating the changes into one peer

requires to check its effect on other partner peers. This approach is implemented into DYCHOR.

In (Preda et al. 2015; Fdhila et al. 2015), both approaches study the evolution that might arise at the peers side. The authors propagate the change from one peer to other partners. The work proposed in (Fdhila et al. 2015) applies to Business Process Management (BPM) domain and Service Oriented Architecture (SOA). It describes service choreographies using tree-based model. The authors consider some changes such as "replace, delete, update, and insert" of behavioural fragments.

(Preda et al. 2015) define a new language referred to as DIOC for programming distributed applications that are free from deadlocks and races by construction. The semantics of DIOC language relies on labelled transition systems. The approach given in (Preda et al. 2015) enables to update fragment of codes of distributed peers. This can be specified at the choreography level where blocks of code that can be updated dynamically must be tagged using "scope". These "scope" blocks have to be known a priori when describing the choreography. The solutions given in (Preda et al. 2015; Roohi and Salaün 2011; Jureta et al. 2007) deal with run-time evolution.

To the best of our knowledge, we are the first who verify the evolution at the CP level such that its realisability must be preserved. Furthermore, we have no restriction on the application domain, yet we use a formal model which can be applied for design, verification, and implementation of different distributed systems, *e.g.* Web services, concurrent systems, Cyber Physical Systems, etc. Our result applies also for asynchronously communicating systems as far as these are synchronisable with no restriction on the buffer size.

7. CONCLUSION AND PERSPECTIVES

In this paper, we presented a preliminary solution for correct evolution of distributed system for which their interaction is described with a conversation protocol. We proposed a language which enables one to incrementally design distributed system that can be updated over time such that their realisability is preserved while applying and composing evolution operators.

In the near future we aim at formalising all properties and composition operators used in this paper. We intend to prove that our evolution operators and their properties preserve CP realisability. Our conjectures have also to be formally proved. We aim also at defining looping and parallel operators as well as extending our language with new operators for messages broadcast and multicast. Lastly, we are using theorem proving techniques in order to prove the correctness of CP evolution. Based on proof-based techniques, we aim at handling any number of peers and exchanged messages such that scalability is ensured.

REFERENCES

Basu, S. and T. Bultan (2016). Automated Choreography Repair. In *Proc. of FASE'16*, Volume 9633 of *LNCS*, pp. 13–30. Springer.

Basu, S., T. Bultan, and M. Ouederni (2012). Deciding Choreography Realizability. In *Proc. of POPL'12*, pp. 191–202. ACM.

Ben-Abdallah, H. and S. Leue (1997). Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts. In *Proc of TACAS'97*, Volume 1217 of *Lecture Notes in Computer Science*, pp. 259–274. Springer.

Brand, D. and P. Zafiropulo (1983). On Communicating Finite-State Machines. *J. ACM 30*(2), 323–342.

Bultan, T. (2006). Modeling interactions of web software. In *Proc. of WWV'06*, pp. 45–52. IEEE.

Farah, Z., Y. Ait-Ameur, M. Ouederni, and K. Tari (2016). A Correct-by-Construction Model for Asynchronously Communicating Systems. *International Journal on Software Tools for Technology Transfer*. To appear.

Fdhila, W., C. Indiono, S. Rinderle-Ma, and M. Reichert (2015). Dealing with change in process choreographies: Design and implementation of propagation algorithms. *Information systems 49*, 1–24.

Güdemann, M., G. Salaün, and M. Ouederni (2012). Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In *Proc. of ATVA'12*, Volume 7561 of *LNCS*, pp. 238–253. Springer.

Hopcroft, J. E. and J. D. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Addison Wesley.

Jureta, I. J., S. Faulkner, and P. Thiran (2007). *Dynamic Requirements Specification for adaptable and open Service-oriented Systems*. Springer.

Leite, L. A., G. A. Oliva, G. M. Nogueira, M. A. Gerosa, F. Kon, and D. S. Milojicic (2013). A Systematic Literature Review of Service Choreography Adaptation. *Service Oriented Computing and Applications 7*(3), 199–216.

Medvidovic, N. (1996). ADLs and dynamic Architecture Changes. In *Proc. of SIGSOFT'96 workshops*, pp. 24–27. ACM.

Preda, M. D., M. Gabbrielli, S. Giallorenzo, I. Lanese, and J. Mauro (2015). Dynamic Choreographies - Safe Runtime Updates of Distributed Applications. In *Proc. of COORDINATION'15*, Volume 9037 of *LNCS*, pp. 67–82. Springer.

Rinderle, S., A. Wombacher, and M. Reichert (2006a). Evolution of process choreographies in DYCHOR. In *Proc. of CoopIS'06*, Volume 4275 of *LNCS*, pp. 273–290. Springer.

Rinderle, S., A. Wombacher, and M. Reichert (2006b). On the Controlled Evolution of Process Choreographies. In *Proc. of ICDE'06*, pp. 124–124. IEEE.

Roohi, N. and G. Salaün (2011). Realizability and Dynamic Reconfiguration of Chor Specifications. *Informatica (Slovenia) 35*(1), 39–49.

Ryu, S. H., F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul (2008). Supporting the Dynamic Evolution of Web Service Protocols in Service-oriented Architectures. *ACM Transactions on the Web (TWEB) 2*(2), 13.

Wombacher, A. (2009). Alignment of Choreography Changes in BPEL Processes. In *Proc. of SCC'09*, pp. 1–8. IEEE.