Received: 18 January 2019 | Revised: 22 May 2019 | Accepted: 6 June 2019

DOI: 10.1002/smr.2209

SPECIAL ISSUE PAPER

WILEY Software: Evolution and Process

Formal design of scalable conversation protocols using Event-B: Validation, experiments, and benchmarks

Sarah Benyagoub[1,2] | Yamine Aït-Ameur[2] | Meriem Ouederni[2] | Atif Mashkoor[3,4] | Ahmed Medeghri[1]

[1]University Abd El Hamid Ibn Badis, Mostaganem, Algeria

[2]IRIT-INP, University of Toulouse, Toulouse, France

[3]Software Competence Center, Hagenberg GmbH, Hagenberg, Austria

[4]Johannes Kepler University, Linz, Austria

Correspondence
Sarah Benyagoub, IRIT-INP, University of Toulouse, Toulouse, France.
Email: sarah.benyagoub@enseeiht.fr

Abstract

Contemporary interaction-based complex systems are often built by reusing existing distributed peers, which have to coordinate with each other to fulfill the client, system, and environment requirements. In this paper, we address the design of distributed systems composed of peers (state-transitions systems) communicating through message exchanges. We consider choreographies as the formal model, allowing a developer to describe and specify peers coordination as a set of conversations; ie, all sequences of messages exchanged between the communicating peers. Proceeding this way requires building neither the individual peers nor their composition as they may be obtained by the choreography projection. The correctness of the preservation of such messages exchanges by each peer obtained after projection is a key issue, known as the realizability problem. Checking choreography realizability is mandatory to build third-party applications with no coordination error, eg, absence of deadlocks, missing messages, and erroneous messaging order. In our previous work, we have proposed a set of composition operators, allowing designers to build realizable choreographies that are represented by conversation protocols (CPs). In this work, realizability is guaranteed by construction. We rely on the correct-by-construction Event-B method to prove that each CP constructed using our operators is realizable. In this paper, we show how our approach applies and scales to a set of use cases borrowed from the literature and used by the research community. We also show that our approach allows to detect failures and failure recovery in case realizability does not hold.

KEYWORDS

choreography realizability, conversation protocols, correct by construction, distributed systems, Event-B, proof and refinement-based methods

1 | INTRODUCTION

In recent years, software systems have moved from monolithic applications to decentralized, heterogeneous, and independent subsystems, which are designed and executed with different organizations in a distributed setting. In general, these communication systems are made of peers that communicate through message exchanges; ie, sending and receiving messages are exchanged between peers taking part to the communication. It is mandatory for peers to coordinate with each other. The idea of coordination consists in allowing a designer to view the third-party system interaction as a centralized one.

In a top-down design of distributed systems, the interaction among peers is usually defined using a global specification called choreography or conversation protocols (CPs). This model specifies behavioral interactions among peers by describing the allowed sequences of sent messages. A main concern that is already addressed by the research community is the verification of CP realizability. It refers to the verification whether there exists a set of peers where their composition generates the same sequences of sent messages as specified by the CP. Benyagoub et al,[1-3] this realizability problem is undecidable in the most general setting[4] due to the possibly ever-increasing queuing mechanism and unbounded buffers. The recent work of Basu et al[5] proposed conditions for verifying that 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 realizability issue for a subclass of asynchronously communicating peers, namely, the synchronizable systems; ie, the system composed of interacting peers behaves equivalently by applying the synchronous or asynchronous communication. A CP is realizable if there exists a set of peers implementing that CP; ie, they send messages to each other in the same order as the CP, and their composition is synchronizable. In this paper,[5] the full checking of the CP realizability applies the following steps: (a) peers projection from the CP; (b) checking synchronizability; and (c) checking equivalence between the CP and its distributed system. The work presented in the paper[5] relies on model checking for systems with reasonable sizes (ie, number of states, transitions, and communicating peers). By doing so, the existing verification methods suggest a posteriori realizability checking.

Recently,[1] we introduced a refinement and proof-based approach where the CP realizability is guaranteed at the CP definition level. It has identified some conditions, allowing to assert that a CP is realizable without requiring to build the synchronous or asynchronous peers compositions. This approach suggests a priori realizability checking. A set of operators (sequence, choice, and loop) are composed to build correct-by-construction realizable CPs. The approach has been proved using the Event-B method[6] on the RODIN platform.[7]

The objective of this paper is to present the validation of our proposed approach[1] using a benchmark of several realizable and nonrealizable real-world case studies from the literature. We also focus on the scalability of the approach in the case of complex CPs. A reparation strategy is also discussed.

The remainder of this paper is structured as follows. Section 2 gives a brief overview of the Event-B method. Section 3 introduces the formal definitions and the background on which our proposal relies. Section 4 recalls the set of composition operators for building realizable CPs. Section 5 shows an excerpt of the sequence, choice, and loop composition development built using the RODIN platform. Section 6 discusses the validation of our proposed approach applying on a set of standard benchmark case studies. Then, an assessment of the whole approach is given in Section 7. Section 8 overviews the related work. Finally, we conclude the paper along with perspectives in Section 9.

2 | EVENT-B

Event-B[6] is a step-wise formal development method. Each Event-B model encodes a state-transition system where variables represent the state and events represent transitions from one state to another. Set theory and first-order logic describe the manipulated concepts. Event-B is based on the idea of step-wise model development with refining an initial model by gradually adding design decisions. A set of proof obligations (POs), based on the weakest precondition calculus, is associated with each machine. The correctness of the development is guaranteed by proving these POs. The refinement capability of Event-B makes it possible to decompose a model (thus a transition system) into another transition system with more design decisions while moving from an abstract level to a concrete one. The refinement process preserves the proved properties, and therefore, it is not necessary to reprove them in the refined transition system (which is usually more complex). The RODIN platform[7] is an eclipse-based integrated development environment (IDE) for developing Event-B models. It is equipped with a set of provers for discharging the generated POs. In addition, we can also use ProB[8] as an animator and model checker for analyzing the developed Event-B models in the RODIN development environment.

2.1 | Modeling

The Event-B language uses set theory and first-order logic. It has two main components: context and machine, to characterize systems. A context describes the static structure of a system using carrier sets s, constants c, axioms $A(s,c)$, and theorems $T_d(s,c)$, and a machine describes the dynamic structure of a system using variables v, invariants $I(s,c,v)$, theorems $T_m(s,c,v)$, variants $V(s,c,v)$, and events evt (see Table 1). A list of events can be used to model possible system behavior to modify state variables by providing appropriate guards in a machine. A set of invariants and theorems can be used to represent relevant properties to check the correctness of the formalized behavior. To define the convergence properties, variants can be used.

2.2 | Refinement

Refinement decomposes a model (thus a transition system) into another transition system containing more design decisions while moving from an abstract level to a concrete one. It supports the modeling of a system gradually by introducing safety properties at various refinement levels. New variables and new events may be introduced. These refinements preserve the relation between the refining model and the refined one while

TABLE 1 Model structure

Context	**Machine**
ctxt_id_2	*machine _id _2*
Extends	**Refines**
ctxt _id _1	*machine _id _1*
SETS	**SEES**
s	*ctxt _id _2*
Constants	**Variables**
c	*v*
Axioms	**Invariants**
$A(s,c)$	$I(s,c,v)$
THEOREMS	**THEOREMS**
$T_c(s,c)$	$T_m(s,c,v)$
END	**VARIANT**
	$V(s,c,v)$
	EVENTS
	Event*evt*
	any*x*
	where*G*(*s*,*c*,*v*,*x*)
	then
	$v{:\mid}BA(s,c,v,x,v')$
	end
	END

introducing new events and variables to specify more concrete behavior of the system. The defined abstract and concrete state variables are linked by introducing gluing invariants.

2.3 | Proof obligations and proof process

To verify the correctness of an Event-B model (machine or refinement), the generated POs (issued from the calculus of substitutions) need to be proved. A proof system allows to prove the POs. The main POs are listed in Table 2, in which the prime notation is used to denote the value of a variable after an event is triggered. These POs require to demonstrate that the theorems hold, each event preserves the invariant (inductive), each event can be triggered (feasibility), and if a variant is declared, it shall decrease. The proof process associated with these POs is inductive. The initialization event and other model events must be preserved by the given invariants. Regarding refinement, two more relevant POs need to be discharged. First, the simulation PO to show that the new modified action in the refined event is not contradictory to the abstract action and the concrete event simulates the corresponding abstract event. Second, in the refined events, we can strengthen the abstract guards to specify more concrete conditions.

TABLE 2 Proof obligations

Theorems	$A(s,c)\Rightarrow T_c(s,c)$
	$A(s,c)\wedge I(s,c,v)\Rightarrow T_m(s,c,v)$
Invariant	$A(s,c)\wedge I(s,c,v)\wedge G(s,c,v,x)\wedge BA(s,c,v,x,v')$
preservation	$\Rightarrow I(s,c,v')$
Event	$A(s,c)\wedge I(s,c,v)\wedge G(s,c,v,x)$
feasibility	$\Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant	$A(s,c)\wedge I(s,c,v)\wedge G(s,c,v,x)\wedge BA(s,c,v,x,v')$
progress	$\Rightarrow V(s,c,v')<V(s,c,v)$

3 | BACKGROUND AND NOTATIONS

We use labeled transition systems (LTSs) as models for CP and peers.

Definition 1. (Peer) A peer is an LTS $P = < S_i, s_i^0, \sum_i, T_i >$ 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 and receive messages, and a unit set of internal action. $T \subseteq S \times \Sigma \times S$ is the 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 for representing internal activities. A transition is represented as $s \xrightarrow{l} s'$ where $l \in \Sigma$. Notice that we refer to a state $s^f \in S$ as final if there is no outgoing transition from s^f.

Definition 2. (Conversation protocol *CP*) A conversation protocol *CP* for a set of peers $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ is an LTS $CP = < S_{CP}, s_{CP}^0, L_{CP}, T_{CP} >$ where S_{CP} is a finite set of states and $s_{CP}^0 \in S_{CP}$ is the initial state; L_{CP} is a set of labels where a label $l \in L_{CP}$ is denoted as $m^{\mathcal{P}_i,\mathcal{P}_j}$ 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: $\forall \{m^{\mathcal{P}_i,\mathcal{P}_j}, m'^{\mathcal{P}_i',\mathcal{P}_j'}\} \subseteq L_{CP} : m = m' \rightarrow \mathcal{P}_i = \mathcal{P}_i' \wedge \mathcal{P}_j = \mathcal{P}_j'$. *CP* s.

Definition 3. (Basic conversation protocol CP_b)Let $CP_b = < S_{CP}, s_{CP}^0, L_{CP}, T_{CP} >$ be a given conversation protocol in *CP*. *CP* is a basic conversation protocol if and only if $T_{CP_b} = \left\{ s_{CP_b} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s'_{CP_b} \right\}$ (ie, a conversation protocol with a single transition).CP_b and *CPB* denote, respectively, a basic conversation protocol and set of basic conversation protocols.

In the remainder of this paper, we denote

- a transition $t \in T_{CP_b}$ as $s_{CP_b} \xrightarrow{m^{\mathcal{P}_i,\mathcal{P}_j}} s'_{CP_b}$ where s_{CP_b} and s'_{CP_b} are source and target states and $m^{\mathcal{P}_i,\mathcal{P}_j}$ is the transition label, and
- we refer to the set of final states as $S^f_{CP_b}$ where the system can terminate its execution.

Definition 4. (Projection) *CP* projection operation, noted $\downarrow CP$, produces a set of peers LTSs $\mathcal{P}_i = < S_i, s_i^0, \sum_i, T_i >$ obtained by replacing in $CP = < S_{CP}, s_{CP}^0, L_{CP}, T_{CP} >$ (see Definition 2) each label $(\mathcal{P}_j, m, \mathcal{P}_k) \in L_{CP}$ with $m?$ if $j=i$ and with $m!$ if $k=i$, otherwise with τ (internal action). Finally, this τ transitions can be removed by applying the standard minimization algorithm.[9]

Definition 5. (Synchronous and asynchronous composition) $Sys_{sync}(\downarrow CP)$ and $Sys_{sync}(\downarrow CP)$ represent the synchronous and asynchronous compositions, respectively, of the peers of $\downarrow CP$ obtained by the projection of *CP*.

Definition 6. (Realizability) This realizability definition is borrowed from Basu et al.[5] It is decomposed into the conjunction of three subproperties:

- Equivalence (≡). $CP \equiv Sys_{sync}(\downarrow CP)$ if *CP* and $Sys_{sync}(\downarrow CP)$ have equal sequences for exchanged messages, ie, trace equivalence.
- Synchronizability. The synchronous system $Sys_{sync}(\downarrow CP)$ and asynchronous system $Sys_{sync}(\downarrow CP)$ are synchronizable if the system behavior is same in both synchronous and asynchronous communications.
- Well formedness (WF). $Sys_{sync}(\downarrow CP)$ is well formed if all the running unbounded queues of the asynchronous system become empty at the end of the asynchronous composition.According to Basu et al,[5] for each deterministic *CP*, the realizability property is guaranteed if the conjunction of the previous properties holds. We write**Realizability** = *Equivalence* ∧ *Synchronizability* ∧ *WF*. We define *R* as a set of realizable *CP*. A proof of correctness of global system realizability is given in Basu et al.[5] This proof has been formalized using Event-B in Farah et al.[10]

3.0.0.1 | Remark

The proofs provided by Basu et al[5] and formalized in Event-B by Farah et al[10] require to build the projections $Sys_{sync}(\downarrow CP)$, $Sys_{sync}(\downarrow CP)$, and their compositions. To avoid building compositions for the verification of realizablity, we have set up an incremental verification of realizability using a correct-by-construction approach while building *CP*. Sufficient conditions have been identified at the *CP* level and do not require to build the projection nor to build the peers synchronous and asynchronous compositions. The approach is scalable; it has been formalized using Event-B. All details of the approach can be found in our previous work.[1]

4 | A LANGUAGE FOR REALIZABLE CONVERSATION PROTOCOLS

This section summarizes the results of our previous work.[1] It gives definitions of the set of operators allowing to build correct-by- construction realizable *CP* and identifies the sufficient conditions to build such *CP*.

4.1 | *CP* composition operators

The defined composition operators are $\otimes_{(\gg, s^f_{CP})}$ (sequence), $\otimes_{(+, s^f_{CP})}$ (branching), and $\otimes_{(\circlearrowleft, s^f_{CP})}$ (iteration) where $s^f_{CP} \in S^f_{CP}$. Each expression of the form $\otimes_{(op,\ s^f_{CP})}(CP,\ CP_b)$ assumes that the initial state of CP_b is fused with the final state s^f_{CP}. Informally, we can say that CP_b is appended to CP at state s^f_{CP}.

Definition 7. Sequential composition $\otimes_{(\gg, s^f_{CP})}$. Given a $CP \in CP$, a state $s_{CP} \in S^f_{CP}$, and a CP_b in CPB where $T_{CP_b} = \left\{ s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b} \right\}$, the sequential composition $CP_{\gg} = \otimes_{(\gg, s_{CP})}(CP,\ CP_b)$ is defined as

- $S_{CP_{\gg}} = S_{CP} \cup \left\{ s'_{CP_b} \mid s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b} \in T_{CP_b} \right\}$,
- $L_{CP_{\gg}} = L_{CP} \cup \{ l_{CP_b} \}$,
- $T_{CP_{\gg}} = T_{CP} \cup \left\{ s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b} \right\}$,
- $S^f_{CP_{\gg}} = (S^f_{CP}\ \{s_{CP}\}) \cup \left\{ s'_{CP_b} \right\}$.

  Here, CP_b must be executed after CP starting from s_{CP}.

Definition 8. Choice composition $\otimes_{(+, s^f_{CP})}$. Given a $CP \in CP$, a state $s_{CP} \in S^f_{CP}$, a set $\{CP_{bi} \in CPB | i = [1..n],\ n \in \mathbb{N}\}$ such that $\forall\ T_{CP_{bi}},\ T_{CP_{bi}} = \left\{ s_{CP} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}} \right\}$, the branching composition $CP_{+} = \otimes_{(+, s_{CP})}(CP, \{CP_{bi}\})$ is defined as

- $S_{CP_{+}} = S_{CP} \cup \left\{ s'_{CP_1},\ \ldots,\ s'_{CP_{bn}} \mid s_{CP} \xrightarrow{l_{CP_{bi}}} s'_{CP_{bi}} \in T_{CP_{bi}} \right\}$,
- $L_{CP_{+}} = L_{CP} \cup \{ l_{CP_{bi}},\ \ldots,\ l_{CP_{bn}} \}$,
- $T_{CP_{+}} = T_{CP} \cup \left\{ s_{CP} \xrightarrow{l_{CP_{b1}}} s'_{CP_{b1}},\ \ldots,\ s_{CP} \xrightarrow{l_{CP_{bn}}} s'_{CP_{bn}} \right\}$,
- $S^f_{CP_{+}} = (S^f_{CP}\ \{s_{CP}\}) \cup \left\{ s'_{CP_{b1}},\ \ldots,\ s'_{CP_{bn}} \right\}$.

  Here, CP must be executed before $\{CP_{bi}\}$, and there is a choice between all $\{CP_{bi}\}$ at s_{CP}.

Definition 9. Loop composition $\otimes_{(\circlearrowleft, s^f_{CP})}$. Given a $CP \in CP$, a state $s_{CP} \in S^f_{CP}$, and a $CP_b \in CPB$, with $T_{CP_b} = \{ s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b} \}$ and $s'_{CP_b} \in S_{CP}$, then the loop composition $CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s_{CP})}(CP,\ CP_b)$ is defined as

- $S_{CP_{\circlearrowleft}} = S_{CP}$,
- $L_{CP_{\circlearrowleft}} = L_{CP} \cup \{ l_{CP_b} \}$,
- $T_{CP_{\circlearrowleft}} = T_{CP} \cup \left\{ s_{CP} \xrightarrow{l_{CP_b}} s'_{CP_b} \right\}$,
- $S^f_{CP_{\circlearrowleft}} = S^f_{CP}$. The condition $s'_{CP_b} \in S_{CP}$ means that the target state of CP_b is a state of CP. It defines a cycle in the built $CP_{\circlearrowleft}$, thus a loop and an iteration. The final states remain unchanged.

4.2 | Realizable-by-construction *CP*

As mentioned previously, our intention is to avoid a posteriori global verification of realizability by building synchronous and asynchronous peers compositions. We set up an incremental verification of realizability using a correct-by-construction approach. Building *CP* using the aforementioned operators does not yet guarantee its realizability. Indeed, the definitions of the previous operators require additional sufficient conditions to guarantee realizability.

4.2.1 | Sufficient conditions for correct-by-construction realizable *CP*

We recall the definitions of the identified sufficient conditions (ie, Conditions 1, 2, and 3) entailing realizability.

These conditions are based on the semantics of the messages ordering and exchange.

Condition 1. (Deterministic choice (DC)) A given CP is deterministic, denoted $CP \in$ DC, if and only if $\forall s_{CP} \in S_{CP}, \forall sCP' \in S_{CP}, \forall sCP'' \in S_{CP}$, $\nexists \left\{ s_{CP} \xrightarrow{m^{P_i,P_j}} s'_{CP},\ \text{and}\ s_{CP} \xrightarrow{m^{P_i,P_j}} s''_{CP} \right\} \subseteq T_{CP}$, such that $sCP' \neq sCP''$.

Condition 2 (Parallel-choice freeness (PCF)) 2. Let PCF be the set of *CP*s free of parallel choice. Then, $CP \in$PCF if $\forall s_{CP} \in S_{CP}, \forall s'_{CP} \in S_{CP}, \forall s''_{CP} \in S_{CP}, \nexists \{s_{CP} \xrightarrow{m^{P_i,P_j}} s'_{CP}$, and $s_{CP} \xrightarrow{m'^{P_k,P_q}} s''_{CP}\} \subseteq T_{CP}$ such that $P_i \neq P_k$ and $sCP' \neq sCP''$.

This condition imposes that when more than two transitions exit a given state, the PCF condition imposes that all the sending peers are the same. In other words, it defines a form of broadcasting.

Condition 3 (Independent sequences freeness (ISeqF)) 3. The condition related to two consecutive message exchanges at the CP level is declined in two conditions corresponding to two orders of sending messages. According to Finkel et al,[11] the first condition preserves the order of both sending and receiving messages by defining a ring structure. The second condition preserves the order of sending messages, and no constraint is imposed on the messages receiving order.

- Send-receive messages order preservation. Let ISeqF be the set of *CP* free from independent sequences. Then, $CP \in$ISeqF if and only if

$$\forall s_{CP} \in S_{CP}, \forall s'_{CP} \in S_{CP}, \forall s''_{CP} \in S_{CP}, \nexists \left\{ s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP}, \text{ and } s'_{CP} \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s''_{CP} \right\} \subseteq T_{CP} \text{ such that } \mathcal{P}_j \neq \mathcal{P}_k.$$

- This condition preserves the order of sending and receiving messages. It enforces a ring communication peers topology.
- Sending messages order preservation. Let ISeqF be the set of *CP* free from independent sequences. Then, $CP \in$ISeqF if and only if

$$\forall s_{CP} \in S_{CP}, \forall s'_{CP} \in S_{CP}, \forall s''_{CP} \in S_{CP}, \nexists \left\{ s_{CP} \xrightarrow{m^{\mathcal{P}_i \to \mathcal{P}_j}} s'_{CP}, \text{ and } s'_{CP} \xrightarrow{m'^{\mathcal{P}_k \to \mathcal{P}_q}} s''_{CP} \right\} \subseteq T_{CP} \text{ such that } \mathcal{P}_j \neq \mathcal{P}_k \text{ and } \mathcal{P}_i \neq \mathcal{P}_k.$$

- This condition preserves the order of sending messages.

All these conditions are structural conditions defined at the CP level. They involve conditions neither on the synchronous nor on the asynchronous projections and/or compositions. These sufficient conditions are checked at the *CP* level.

4.2.2 | Realizable-by-construction *CP* theorems

The theorems—established in our previous work[1]—rely on the previously introduced sufficient conditions. They ensure the realizability of a *CP* built incrementally using each of the defined operators. These theorems are defined as follows.

1. Any basic conversation protocol CP_b is realizable.
   - Theorem 1. $\forall CP_b \in CPB, CP_b \in R$
2. The sequence composition operator preserves realizability when ISeqF property holds.
   - Theorem 2. $\forall CP \in CP \forall CP_b \in CPB,\ CP \in R \wedge CP_b \in R \wedge CP_{\gg} = \otimes_{(\gg, s^f_{CP})}(CP, CP_b) \in \text{ISeqF} \rightarrow CP_{\gg} \in R$
3. The choice composition operator preserves realizability when PCF property and ISeqF properties old for each transition if the branch.
   - Theorem 3. $\forall\ CP \in CP\ \forall\ CP_b set = \{CP_{bi} \in CPB | i = [2..n], n \in \mathbb{N}\},\ CP \in R \wedge CP_b set \subseteq R \wedge CP_+ = \otimes_{(+, s^f_{CP})}(CP, CP_b set) \in \text{DC} \wedge CP_+ \in \text{ISeqF} \wedge CP_+ \in \text{PCF} \Rightarrow CP_+ \in R$
4. The loop composition operator preserves realizability ISeqF property holds.
   - Theorem 4. $\forall CP \in CP \forall CP_b \in CPB,\ CP \in R \wedge CP_b \in R \wedge CP_{\circlearrowleft} = \otimes_{(\circlearrowleft, s^f_{CP})}(CP, CP_b) \in \text{ISeqF} \rightarrow\ CP_{\circlearrowleft} \in R$

The correctness of these theorems has already been proven in our previous work[1] using the Event-B method on the RODIN platform.[1] The proof is based on the structural induction. Refinement is used to introduce gradually equivalence, synchronizability and well-formedness properties of Definition 6.

5 | A FORMAL MODEL FOR REALIZABLE *CP*: REFINEMENT-BASED REALIZABILITY

This section presents the Event-B developments modeling the defined composition operators. An Event-B event is associated with each operator allowing to build realizable *CP*. Each event (*Initialization*, *Add_Sequence*, *Add_Choice*, and *Add_Loop*) is guarded by the previously identified sufficient conditions.

The Event-B model is defined as two parts. The context part, presented in Sections 5.1 and 5.2, contains the relevant axioms defining peers, *CP*, etc, together with sufficient conditions. These definitions and conditions are used to define the behavioral part of the model and to prove its

[1]The whole Event-B developments are available at http://yamine.perso.enseeiht.fr/EventB\mathunderscoreRealisability_Models.pdf.

correctness since they provide with hypotheses. The second part of the model (see Section 5.3) describes in Event-B the state transitions systems using the defined set of operators. Finally, the formal development establishing realizability is described in Section 5.4.

5.1 | Basic definitions

The basic definitions of the notions of conversation protocols, peers, messages, etc, and the various sets and relations needed to build our development are given in a single Event-B context presented in Listing 1.

```
Context
Sets
   PEERS,                                  -- set of peers
   MESSAGES,                               -- set of exchanged messages
   CP_STATES,                              -- set of possible states for CP
Constants
   CPs_B,                                  -- constant set containing of basic CP transitions
   SOURCE_STATE,                           -- function returning CP source state
   DESTINATION_STATE,                      -- function returning CP target state
   LABEL,                                  -- function returning CP label
   PEER_SOURCE,                            -- function returning CP source peer
   LAST_SENDER_RECEIVER_PEERS,             -- function returning CP last sender and receiver peers
   NDC,                                    -- constant set of non deterministic CPs
   DC,                                     -- constant set of deterministic CPs
   ISeqF,                                  -- constant set of CPs satisfying the ISeqF condition
   PCF,                                    -- constant set of CPs satisfying the PCF condition
   ...
Axioms
   -- Basic conversation protocols definition
 axm1_CP : CPs_B ⊆ CP_STATES × PEERS × MESSAGES × PEERS × CP_STATES × ℕ
 ...
```

Listing 1: Definition of sufficient conditions

*axm*1_*CP* defines the notion of basic conversation protocols defined as a set of tuples of the form (source state, sending peer, exchanged message, receiving peer, target state, and index) where the index is a label for the defined transition. This label has been added to ease the manipulation of transitions in the proof process.

5.2 | Sufficient conditions

Note that all set definitions of the sets for nondeterministic *CP*, ISeqF, and PCF properties are given using an equivalence relationship decomposed into two implications. We have chosen this decomposition for proof purpose.

- Deterministic choice (DC) is defined from nondeterministic choice (NDC) transitions in axioms *axm*2_*Cond*1, *axm*3_*Cond*1_1, and *axm*3_*Cond*1_2 of Listing 2. Then, the deterministic transitions DC are obtained by subtraction in axiom *axm*4_*Cond*1.

```
-- Determinstic CP definition DC
axm2_Cond1 : NDC ⊆ CPs_B
axm3_Cond1_1 : ∀ Trans1, Trans2· (
                        Trans1 ∈ CPs_B ∧ Trans2 ∈ CPs_B ∧
                        SOURCE_STATE(Trans1) = SOURCE_STATE(Trans2) ∧
                        LABEL(Trans1) = LABEL(Trans2) ∧
                        DESTINATION_STATE(Trans1) ≠ DESTINATION_STATE(Trans2)
                        )
                        ⇒
                        {Trans1, Trans2} ⊆ NDC

 axm3_Cond1_2 : ∀ Trans1 . Trans1 ∈ NDC
                ⇒
                ∃ Trans2 ∈ NDC .
                (
                SOURCE_STATE(Trans1) = SOURCE_STATE(Trans2) ∧
                LABEL(Trans1) = LABEL(Trans2) ∧
                DESTINATION_STATE(Trans1) ≠ DESTINATION_STATE(Trans2)
                )

axm4_Cond1 : DC = CPs_B \ NDC
```

Listing 2: Definition of Deterministic CP condition

- The definition of ISeqF is given by axioms *axm5_Cond2*, *axm6_Cond2_1*, and *axm6_Cond2_2* of Listing 3. It asserts that the source peer *PEER_SOURCE*(*cp_b*) of a *cp_b* is either the last sender or receiver peer of *cp_b*.

$$
\begin{array}{l}
\textbf{-- Independent sequence freeness definition ISEQF} \\
axm5\_Cond2 : \ ISeqF \subseteq CPs\_B \\
axm6\_Cond2\_1 : \forall\, cp\_b \cdot (cp\_b \in CPs\_B \wedge \\
\qquad PEER\_SOURCE(cp\_b) \in LAST\_SENDER\_RECEIVER\_PEERS(SOURCE\_STATE(cp\_b)) \\
\qquad\qquad) \\
\qquad \Rightarrow \\
\qquad \{cp\_b\} \subseteq ISeqF \\
\\
axm6\_Cond2\_2 : \ \forall\, cp\_b \,.\, cp\_b \in ISeqF \\
\qquad \Rightarrow \\
\qquad PEER\_SOURCE(cp\_b) \in LAST\_SENDER\_RECEIVER\_PEERS(SOURCE\_STATE(cp\_b)))
\end{array}
$$

Listing 3: Definition of ISeqF condition

- Similarly, in order to define the PCF property, in axioms *axm7_Cond3*, *axm8_Cond3_1*, and *axm8_Cond3_2* of Listing 4, the sender peers *PEER_SOURCE*(*Trans*) of the transitions involved in a branch are compared.

$$
\begin{array}{l}
\textbf{-- Parallel Choice freeness PCF} \\
axm7\_Cond3 : \ PCF \in CP\_STATES \rightarrow \mathbb{P}(CPs\_B) \\
axm8\_Cond3\_1 : \forall\, Trans1,\ Trans2 \,.\, (\\
\qquad Trans1 \in CPs\_B \ \wedge \ Trans2 \in CPs\_B \wedge \\
\qquad SOURCE\_STATE(Trans1) = SOURCE\_STATE(Trans2) \wedge \\
\qquad PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \wedge \\
\qquad DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2) \\
\qquad) \\
\qquad \Rightarrow \\
\qquad \{Trans1, Trans2\} \subseteq PCF(SOURCE\_STATE(Trans1)) \\
\\
axm8\_Cond3\_2 : \ \forall\, Trans1,\ Trans2,\ s \,.\, (\\
\qquad s \in CP\_STATES \wedge \\
\qquad Trans1 \in PCF(SOURCE\_STATE(Trans1)) \wedge \\
\qquad Trans2 \in PCF(SOURCE\_STATE(Trans1)) \wedge \\
\qquad Trans1 \neq Trans2 \wedge \\
\qquad s = SOURCE\_STATE(Trans1) \wedge \\
\qquad s = SOURCE\_STATE(Trans2) \\
\qquad\quad) \\
\qquad \Rightarrow \\
\qquad PEER\_SOURCE(Trans1) = PEER\_SOURCE(Trans2) \wedge \\
\qquad DESTINATION\_STATE(Trans1) \neq DESTINATION\_STATE(Trans2)
\end{array}
$$

Listing 4: Definition of PCF condition

5.3 | Composition operators

Event-B definitions of the events encoding the defined operators (sequence of Definition 7, branch of Definition 8, and loop of Definition 9) are given in this section. Each event is guarded by the defined sufficient conditions such that the CP and the projected peers are composed if and only if the guards hold.

- State and initialization. This event initializes the state variable *BUILT_CP* (in action *act*1 of Listing 5) as an empty initial CP. This variable is updated using the defined composition operator, under the identified sufficient conditions. It also initializes the variables *Prophecy _ of _ Sent _ Messages* and *Number _ of _ send* defining the expected (prophecy) number of exchanged messages and the number of already sent messages. These variables are introduced for proof purposes. When the defined variant is null (0), ie, *Prophecy _ of _ Sent _ Messages*−*Number _ of _ send*=0 (*null*), all the messages should be consumed.

```
Variables
    BUILT_PEERS,
    Number_of_send,
    Prophecy_of_Sent_Messages
Invariants
    inv1 : BUILT_CP ⊆ DC
    inv2 : Number_of_send ∈ ℕ
    inv3 : Prophecy_of_Sent_Messages ∈ ℕ
Variant
    Prophecy_of_Sent_Messages − Number_of_send
Event Initialisation ≙
Any
Where
Then
    act1 : BUILT_CP := ∅
    act2 : Number_of_send := 0
    act3 : Prophecy_of_Sent_Messages :∈ ℕ
    ...
End
```

Listing 5: An excerpt from the initialization event

- *Add_Sequence* operator. Add _Sequence event in Listing 6 corresponds to the sequence operator (Definition 7). It allows to add a basic *CP*, given as a parameter, namely, *Some_cp_b* to the *CP* under construction (using union operation in action *act*1). It also sets up the new final state in action *act*2. This event is triggered only if the relevant conditions hold (guards). In particular, it clearly states that the independent sequence property ISeqF must hold before adding another CP in sequence in guard *grd*3.

```
Event Add_Sequence ≙ Convergent
Any Some_cp_b
Where
    grd1 : Some_cp_b ∈ DC
    grd2 : MESSAGE(Some_cp_b) ≠ End_message
    grd3 : BUILT_CP ≠ ∅ ⇒ Some_cp_b ∈ ISeqF
    grd4 : SOURCE_STATE(Some_cp_b) ∈ CP_Final_states
    ...
Then
    act1 : BUILT_CP := BUILT_CP ∪ {Some_cp_b}
    act2 : CP_Final_states := (CP_Final_states∪
                               {DESTINATION_STATE(Some_cp_b)})\
                               {SOURCE_STATE(Some_cp_b)}
    act3 : Number_of_send := Number_of_send + 1
        ...
End
```

Listing 6: An excerpt from the sequence composition operator

- *Add_Choice* operator. The *Add_Choice* event of Listing 7 encodes the choice operator (Definition 8). It adds a set of deterministic basic CP_b, given as parameter, namely, *Branches*⊆DC. Moreover, any basic *CP*, namely, *branch* belonging to *Branches* with same source state and sender peer, shall satisfy the ISeqF and the PCF properties (*grd*3 and *grd*4) to fulfill the sufficient conditions related to choice operator. *act*1 and *act*2 update the built *CP* and the final states accordingly.

```
Event Add_Choice ≙ Convergent
Any Branches, branch
Where
    grd1 : Branches ⊆ DC
    grd2 : branch ∈ Branches
    grd3 : BUILT_CP ≠ ∅ ⇒ branch ∈ ISeqF
    grd4 : Branches = PCF(SOURCE_STATE(branch))
    grd5 : SOURCE_STATE(branch) ∈ CP_Final_states
    grd6 : MESSAGE(branch) ≠ End_message
    ...
Then
    act1 : BUILT_CP := BUILT_CP ∪ Branches
    act2 : CP_Final_states := (CP_Final_states∪
           BR_CP_FINAL_STATES(SOURCE_STATE(branch)))\{SOURCE_STATE(branch)}
    act3 : Number_of_send := Number_of_send + 1
    ...
End
```

Listing 7: An excerpt from the choice composition operator

- *Add_Loop* operator. *Add_Loop* event of Listing 8 formalizes the loop operator (Definition 9) by adding a loop transition (self loop or cycle loop) given as parameter, namely, *Some_cp_b* to the current *BUILT_CP* in *act*1. Moreover, it requires that the ISeqF property must be held in *grd*2 and *grd*3. Note that the *grd*4 differentiates between the *Add_Loop* event and the *Add_Sequence* event by enforcing the source and destination states of a new transition *Some_cp_b* that is already existing states of the built CP.

```
Event Add_Loop ≙ Convergent
Any Some_cp_b
Where
    grd1 : Some_cp_b ∈ DC
    grd2 : BUILT_CP ≠ ∅ ⇒ Some_cp_b ∈ ISeqF
    grd3 : ∃ Trans.(Trans ∈ BUILT_CP) ∧
                                (
                                    (PEER_SOURCE(Some_cp_b) = PEER_SOURCE(Trans))
                                ∨
                                    (PEER_DESTINATION(Some_cp_b) = PEER_SOURCE(Trans))
                                )
    grd4 : ∃ Trans.(Trans ∈ BUILT_CP) ∧ DESTINATION_STATE(Some_cp_b) = SOURCE_STATE(Trans)
    grd5 : MESSAGE(Some_cp_b) ≠ End
    grd6 : SOURCE_STATE(Some_cp_b) ∈ CP_Final_states
    ...
With
Then
    act1 : BUILT_CP := BUILT_CP ∪ {Some_cp_b}
    act2 : Number_of_send := Number_of_send + 1
    ...
End
```

Listing 8: An excerpt from the loop composition operator

- *Add_End* event.
  When the prophecy variable (declared as a state variable) becomes null (0) *grd*1, the *Add_End* event of the Listing 9 will be enabled such that its execution means the end of the *CP* construction. *Add_End* event adds a new transition exchanging "End" message between two peers to the current built *CP* in *act*1.

```
Event Add_End ≙
Any Some_cp_b
Where
    grd1 : Prophecy_of_Sent_Messages − Number_of_send = 0
    grd2 : SOURCE_STATE(Some_cp_b) ∈ CP_Final_states
    grd3 : MESSAGE(Some_cp_b) = End
    ...
Then
    act1 : BUILT_CP := BUILT_CP ∪ {Some_cp_b}
    ...
End
```

Listing 9: An excerpt from the End composition operator

5.4 | Proving realizability

The above section showed the Event-B models corresponding to the definition of the composition operators we have introduced. These definitions are structural ones; they are guarded by the sufficient conditions we have identified. The realizability preservation property of these operators is not yet established in these definitions.

In order to make this paper self-contained, we borrow from Benyagoub et al[1] the information related to the formal development to prove realizability preservation proof process. Below, we summarize this proof process and describe the whole development for the sequence operator. The development for all the other operators can be downloaded from http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.

5.4.1 | The refinement strategy

Our Event-B formal development heavily relies on refinement. We consider that a finite number of messages is exchanged (we do not deal with infinite number of messages) through the definition of a prophecy variable.[12] We have introduced in refinement different levels the properties

defining realizability, ie, first, equivalence, and then, synchronizability, and finally, well-formedness. These properties are introduced as invariants at different levels, and the proof of invariant preservation is inductive, based on the numbers of exchanged messages.

The development results in the following steps.

Abstract (root) model. It defines the conversation protocols and introduces basic *CP*. Each composition operator is defined as an event, which incrementally builds the final *CP* obtained by introducing a final state. All the built *CP* satisfy an invariant requiring DC (condition 1). This model also declares a prophecy variable [12] as a state variable. This variable defines an arbitrary numbers of exchanged messages and is used to define a variant in order to further prove well formedness. This model corresponds to the Event-B statements presented in Sections 5.2 and 5.3.

First refinement: the synchronous model. The second model is obtained by refining each event (composition operator) to define the synchronous projection. A gluing invariant linking the *CP* to the synchronous projection is introduced. The equivalence property is proved at this level. It is defined as an invariant preserved by all the events encoding a composition operator. This projection represents the synchronous system; it preserves the message exchanges order between peers and hides the asynchronous exchanges.

Second refinement: the asynchronous model. The last model introduces the asynchronous projection. Each event (composition operator) is refined to handle the asynchronous communication. Synchronous and asynchronous projections are linked by another gluing invariant. Sending and receiving actions together with queue handling actions and variant decreasing of the prophecy variable are introduced. At this refinement level, they are necessary to prove synchronizability and well formedness expressed as invariants. The refinement of the synchronous models in an asynchronous model eases the proof process.

At the last refinement level, realizability—defined as the conjunction of equivalence, synchronizability, and well-formedness properties—is proved thanks to invariants preservation and to the inductive proof process handled by Event-B using the RODIN platform.

Next, we show an extract of the development for the sequence operator.

Remark

The equivalence, synchronizability, and well-formedness properties are modeled using relations defined using comprehension. We have introduced the *EQUIV*, *SYNCHRONIZABILITY*, and *WF* sets. A *CP* fulfilling one of these properties shall belong to the corresponding set.

5.4.2 | First refinement: Synchronous model

The objective of the first refinement is to build the synchronous projection corresponding to Definition 5. Here, again, before building this projection, some property definitions are required, in particular for equivalence (≡), denoted *EQUIV* in Event_B models.

Required properties for synchronous projection (cf Listing 10).

```
CONTEXT LTS_SYNC_CONTEXT
EXTENDS LTS_CONTEXT
SETS
   ACTIONS.

CONSTANTS
   CPs_B,
   EQUIV,
   ...

AXIOMS
axm1 : CPs_SYNC_B ⊆
   CP_STATES × ACTIONS × MESSAGES × PEERS × PEERS × ACTIONS × MESSAGES × CP_STATES × ℕ

– – Equivalence of CP and Synchronous projection
axm_1.a : EQUIV ∈ CPs_B ⇻ CPs_SYNC_B
axm_1.a1 : EQUIV = {Trans ↦ S_Trans |
                                    Trans ∈ CPs_B ∧ S_Trans ∈ CPs_SYNC_B ∧
                                    SOURCE_STATE(Trans) = S_SOURCE_STATE(S_Trans) ∧
                                    DESTINATION_STATE(Trans) = S_DESTINATION_STATE(S_Trans) ∧
                                    PEER_SOURCE(Trans) = S_PEER_SOURCE(S_Trans) ∧
                                    PEER_DESTINATION(Trans) = S_PEER_DESTINATION(S_Trans) ∧
                                    MESSAGE(Trans) = S_MESSAGE(S_Trans) ∧
                                    INDEX(Trans) = S_INDEX(S_Trans)
                   }
...
End
```

Listing 10: An excerpt from the synchronous context

The definition of the state-transitions system corresponding to the synchronous projection is given by the set *CPs_SYNC_B* defined by axiom *axm*1 of Listing 10. Actions (send ! and receive ?) are introduced. Then, two other important axioms, namely, *axm*_1.*a* and *axm*_1.*a*1, are given to define the equivalence between a *CP* and its synchronous projection. The *EQUIV* relation is introduced. It characterizes the set of *CP* that is equivalent to their synchronous projection. *axm*_1.*a*1 formalizes the first item of Definition 6 of Section 3.

Invariants
*inv*1 : $BUILT\_SYNCHRONE \subseteq CPs\_SYNC\_B$
*inv*_1.*a* : $\forall Trans \cdot \exists S\_Trans \cdot (Trans \in BUILT\_CP \wedge S\_Trans \in BUILT\_SYNCHRONE \wedge BUILT\_CP \neq \emptyset) \Rightarrow \mathbf{Trans \mapsto S\_Trans \in EQUIV}$
...

Listing 11: An excerpt from the invariants of the synchronous model.

The synchronous projection (cf Listing 12). The first refinement introduces the synchronous projection of the *BUILT_CP* defined by variable *BUILT_SYNCHRONE* in Listing 12.

Listing 11 introduces through invariant *inv*_1.*a*, the equivalence (≡) property of Definition 6. The invariant requires equivalence between a CP and its synchronous projection. Invariant *inv*_1.*a* of Listing 11 describes the equivalence property using the *EQUIV* relation defined in the context of Listing 10. So one part of the realizability property (ie, $CP \equiv Sys_{sync}$) of Definition 6 is already proved at this refinement level.

Event *Initialisation* ≙ ...

Event *Add_Sequence* **Refines** *Add_Sequence* ≙ *Convergent*
Any
S_Some_cp_b, *Some_cp_sync_b*
Where
*grd*1 : $Some\_cp\_sync\_b \in CPs\_SYNC\_B$
*grd*2 : $S\_SOURCE\_STATE(Some\_cp\_sync\_b) \in CP\_Final\_states$
*grd*3 : $BUILT\_CP \neq \emptyset \Rightarrow S\_Some\_cp\_b \in ISeqF$
*grd*4 : $MESSAGE(S\_Some\_cp\_b) \neq End$
*grd*5 : $MESSAGE(S\_Some\_cp\_b) = S\_MESSAGE(Some\_cp\_sync\_b)$
...
With
$Some\_cp\_b : Some\_cp\_b = S\_Some\_cp\_b$
Then
*act*1 : $BUILT\_CP := BUILT\_CP \cup \{S\_Some\_cp\_b\}$
*act*2 : $BUILT\_SYNCHRONE := BUILT\_SYNCHRONE \cup \{Some\_cp\_sync\_b\}$
...
End

Listing 12: An excerpt from the synchronous model.

The event *Add_Sequence* or sequence operator of Listing 12 refines the same event of the root model of Listing 6. It introduces the *BUILT_SYNCHRONE* set corresponding to the synchronous projection as given in Definition 5. Here, again, the *Add_Sequence* applies only if the conditions in the guards hold. The *With* clause provides a witness to glue *Some_cp_b* of *CP* with its synchronous version.

5.4.3 | Second refinement: Asynchronous model

The second refinement introduces the asynchronous projection with sending and receiving peers actions. Well formedness and synchronizability, characterized in Listing 13, remain to be proved in order to complete realizability preservation.

```
CONTEXT LTS_ASYNC_CONTEXT
EXTENDS LTS_SYNC_CONTEXT
SETS
   A_STATES, ...
CONSTANTS
  PEERs_B,
  R_TRACE_B,
  SYNCHRONISABILITY,
  WF,
  ...
AXIOMS
 axm1 : PEERs_B ⊆ (A_STATES × ETIQ × ℕ) ⇸ A_STATES
 axm2 : R_TRACE_B ⊆ CP_STATES × PEERS × MESSAGES × PEERS × CP_STATES × ℕ

−− Synchronisability property
 axm_1.b : SYNCHRONISABILITY ∈ CPs_SYNC_B ⇻ R_TRACE_B
 axm_1.b1 : SYNCHRONISABILITY = {S_Trans ↦ R_Trans |
                                 S_Trans ∈ CPs_SYNC_B ∧ R_Trans ∈ R_TRACE_B ∧
                                 S_INDEX(S_Trans) = R_INDEX(R_Trans) ∧
                                 S_SOURCE_STATE(S_Trans) = R_SOURCE_STATE(R_Trans) ∧
                                 S_PEER_SOURCE(S_Trans) = R_PEER_SOURCE(R_Trans) ∧
                                 S_MESSAGE(S_Trans) = R_MESSAGE(R_Trans) ∧
                                 S_PEER_DESTINATION(S_Trans) = R_PEER_DESTINATION(R_Trans) ∧
                                 S_DESTINATION_STATE(S_Trans) = R_DESTINATION_STATE(R_Trans) ∧
                                 S_INDEX(S_Trans) = R_INDEX(R_Trans)
                                 }

−− Well formedness property
 axm_1.c : WF ∈ A_TRACES → QUEUE
 axm_1.c1 : ∀ A_TR, queue · (A_TR ∈ A_TRACES ∧ queue ∈ QUEUE ∧ queue = ∅)
                          ⇒
                          A_TR ↦ queue ∈ WF
 ...
End
```

Listing 13: An excerpt from the asynchronous context.

The asynchronous Projection (cf Listings 14, 15, 16, and 17). The invariants associated with this model are presented in Listing 14. In particular, the properties of synchronizability, expressed in invariant *inv*_1.*b* used in Definition 6 (*Sync*(Sys_{sync}, Sys_{asynd})), and of well formedness, expressed in invariant *inv*_1.*c* used in Definition 6 (WF(Sys_{async})), are introduced in the invariant of this refinement level. These two properties complete the proof of realizability.

```
Invariants
    inv1 : BUILT_SYNCHRONE ⊆ CP_SYNC_B
    inv2 : REDUCED_TRACE ⊆ R_TRACE_B
    inv3 : A_TRACE ⊆ A_TRACES
    inv_1.b : ∀ S_Trans · ∃ R_Trans · (S_Trans ∈ BUILT_SYNC ∧ R_Trans ∈ REDUCED_TRACE)
                                      ⇒
                                      S_Trans ↦ R_Trans ∈ SYNCHRONISABILITY
    inv_1.c : ∀ A_Trans · (A_Trans ∈ A_TRACES ∧ MESSAGE(Last_cp_trans) = End ∧ A_TRACE ≠ ∅)
                                      ⇒
                                      A_Trans ↦ queue ∈ WF
    inv6 : BUILT_ASYNCHRONE ⊆ PEERs_B
    ...
```

Listing 14: An excerpt from the asynchronous invariants.

At this level, each event corresponding to a composition operator is refined by three events: one to handle sending of messages (*Add_Send*) Listing 15, one for receiving messages (*Add_Receive*) on Listing 16, and a third one (*Add_Sequence_Send−Receive*) in Listing 17 refining the abstract *Add_sequence* event. Sending and receiving events are interleaved in an asynchronous manner. They are defined as follows.

- The *Add_Send* event defines sending of a message. The sending peer sends a messag attached to the queue of the receiving peer (*act*2). The asynchronous trace is updated (*act*1), and the current state is updated to the next state (*act*3). In this asynchronous message exchanges, several sending messages can be triggered if the given guard (*grd*1) of the event is enabled.

```
Event Add_Send ≜
Any
    send, lts_s, lts_d, msg, index
Where
    grd1 : ∃send_st_src, send_st_dest·((lts_s ↦ send_st_src) ∈ A_GS ∧ ((send_st_src ↦
                        (Send ↦ msg ↦ lts_d) ↦ index) ↦ send_st_dest) ∈ CPs_ASYNC_B ∧ ...
    ...
Then
    act1 : A_TRACE := A_TRACE ∪ {Reduces_Trace_states ↦ St_Num ↦
                                 Send ↦ lts_s ↦ msg ↦ lts_d ↦ Reduces_Trace_states ↦
                                 (St_Num + 1) ↦ A_Trace_index
                                }
    act2 : queue, back := queue ∪ {lts_d ↦ msg ↦ back}, back + 1
    act3 : A_GS := A_Next_States({send} ↦ A_GS ↦ queue)
    ...
End
```

Listing 15: An excerpt from the asynchronous model - *Add_Send* event.

- Similar to the previous event, *Add_Receive* consumes the messages available in the queue associated with each peer (*act*2). In the same manner, the asynchronous trace is updated (*act*1). Again, in this asynchronous message exchanges, several receptions of messages can be triggered if the guards of the event are true.

```
Event Add_Receive ≜
Any
    send, receive, lts_s, lts_d, msg, index
Where
    grd1 : queue ≠ ∅
    grd2 : lts_d ↦ msg ↦ front ∈ queue
    grd3 : ∃ receive_st_src, receive_st_dest·(((lts_d ↦ receive_st_src) ∈ A_GS) ∧
                ((receive_st_src ↦ (Receive ↦ msg ↦ lts_s) ↦ index) ↦ receive_st_dest) ∈ CPs_ASYNC_B ∧ ...
    ...
Then
    act1 : A_TRACE := A_TRACE ∪ {Reduces_Trace_states ↦ St_Num ↦
                    Receive ↦ lts_s ↦ msg ↦ lts_d ↦ Reduces_Trace_states ↦ (St_Num + 1) ↦ A_Trace_index
                                }
    act2 : queue := queue \ {lts_d ↦ msg ↦ front}
    ...
End
```

Listing 16: An excerpt from the asynchronous model - *Add_Receive* event.

- Once a pair of send and receive events has been triggered, the event *Add_Sequence_Send−Receive* refining the *Add_Sequence* event records that the emission-reception is completed.
  Traces are updated accordingly by the events; they are used for proving the invariants. Since this event refines the abstract *Add_Sequence* event, it guarantees that the variant decreases.
  Observe that the witnesses given in the *With* clause ensure the conversation protocol and its synchronous and asynchronous projections are correctly glued, it defines the simulation relationship induced by the refinement.

```
Event Add_Sequence_Send − Receive ≜ Convergent
Refines Add_Sequence
Any
    A_Some_cp_b, A_Some_cp_sync_b, Send_cp_async_b, Receive_cp_async_b, R_trace_b
Where
    grd1 : A_MESSAGE(Send_cp_async_b) = A_MESSAGE(Receive_cp_async_b)
    grd2 : ACTION(Receive_cp_async_b) = Receive ∧ ACTION(Send_cp_async_b) = Send
    grd3 : BUILT_CP ≠ ∅ ⇒ A_Some_cp_b ∈ ISeqF
    grd4 : MESSAGE(A_Some_cp_b) = A_MESSAGE(Send_cp_async_b)
    ...
With
        S_Some_cp_b : S_Some_cp_b = A_Some_cp_b,
        Some_cp_sync_b : Some_cp_sync_b = A_Some_cp_sync_b
Then
    act1 : BUILT_CP := BUILT_CP ∪ {A_Some_cp_b}
    act2 : BUILT_SYNCHRONE := BUILT_SYNCHRONE ∪ {A_Some_cp_sync_b}
    act3 : BUILT_ASYNCHRONE := BUILT_ASYNCHRONE ∪ {Send_cp_async_b} ∪ {Receive_cp_async_b}
    act4 : REDUCED_TRACE := REDUCED_TRACE ∪ {R_trace_b}
    ...
End
```

Listing 17: An excerpt from the asynchronous model - *Add_Sequence* event.

The same development chain has been set up for other operators. The whole Event-B development has been proved using the RODIN platform.

6 | CASE STUDIES

In our previous work,[1] we have given proofs of the presented approach. In this paper, our objective is to show that the approach is not only verifiable but also valid. Indeed, we have taken several benchmarks from the research area literature related to the design of distributed systems based on realizability property. All these benchmarks have been used to validate our Event-B model. They have been defined as instances of the generic model summarized in the previous section. These instances are defined in Event-B by defining set extensions of the deferred (arbitrary) sets used in the generic model and by providing witnesses to the parameters of the *ANY* Event-B clause used in each event defining an operator.

This section is divided in two parts. First, in order to explain how instantiation of the generic model is set up, we give a detailed description of the instantiation process using a realizable case study in Section 6.1 and nonrealizable case study in Section 6.2. Then, in a second part, we discuss the obtained results for other selected benchmark case studies. Please note that the instantiation of the generic model is checked using the RODIN platform and the ProB animator.

6.1 | A realizable CP: The controlled evolution of process choreographies

Description. According to Rinderle et al,[13] the CP behavior is depicted in Figure 1 and describes the following scenario. "Initially (state s_0), the accounting department peer (A) receives an order message sent by the buyer peer (B) (state s_1). Then, this message is forwarded to the logistics department peer (L) via a deliver message (state s_2). Next, the logistics department peer answers with a *Deliver_Conf* message (state s_3). When the accounting department peer (A) receives this message, it forwards it to the buyer via a delivery message (state s_4). At this level (state s_4), an alternative holds. Indeed, the accounting department peer (A) may receive a *Get_Status* message sent by the buyer (B) peer (state s_5), followed by an invocation of the logistics status using the *Get_StatusL* message (state s_7). Next, two answer messages (Status and StatusL messages) are sent as answers by logistics department (L) peer (state s_9) and by the accounting department (A) peer (state s_4). At this level (state s_4), this process may be iterated. Alternatively, it is possible to terminate the accounting as well as the logistics. A termination message is initiated by the buyer (B) peer (state s_6) sent to the accounting department (A) peer, which forwards it to the logistics (L) peer (state s_8). At this state (s_8), both peers terminate."

Following Definition 4, the communicating peers are obtained from the *CP* by projection. Figure 2 depicts the accounting department (A), the buyer (B), and the logistics department (L).

Let us check realizability of this case study using our approach. The first step consists in identifying the set of basic conversation protocolsand initializing the currently built CP as an empty set as follows.

- $CP=\varnothing$

Basic transitions

$$cp_{b0} = s_0 \xrightarrow{Order^{B\to A}} s_1,\ cp_{b1} = s_1 \xrightarrow{Deliver^{A\to L}} s_2,\ cp_{b2} = s_2 \xrightarrow{Deliver-Conf^{L\to A}} s_3,\ cp_{b3} = s_3 \xrightarrow{Delivery^{A\to B}} s_4,$$

$$cp_{b4} = s_4 \xrightarrow{Get-Status^{B\to A}} s_5,\ cp_{b5} = s_4 \xrightarrow{Terminate^{B\to A}} s_6,\ cp_{b6} = s_5 \xrightarrow{Get-StatusL^{A\to L}} s_7,\ cp_{b7} = s_6 \xrightarrow{TerminateL^{A\to L}} s_8,$$

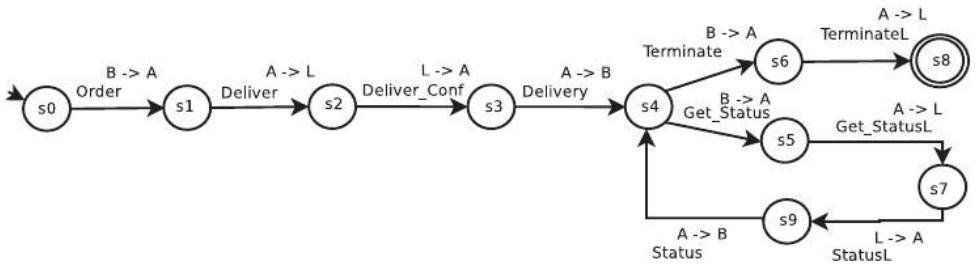

FIGURE 1 Simple procurement process within a virtual company

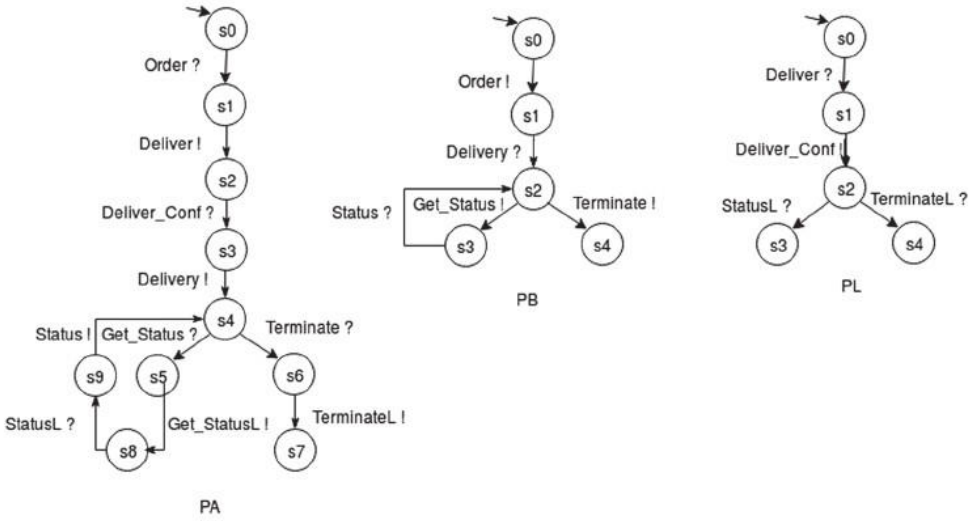

FIGURE 2 Projected peers of a simple procurement process within a virtual company

$cp_{b8} = s_7 \overset{StatusL^{L\to A}}{\longrightarrow} s_9$, $cp_{b9} = s_9 \overset{Status^{A\to B}}{\longrightarrow} s_4$.

Next step consists of applying the defined operators building a sequence of compositions leading to build the CP of Figure 1. Then, we apply the different operators (sequence, branch, and loop) defined previously on a set of basic conversation protocols. These operators are applied only if the sufficient conditions associated with each composition operator hold. The following sequence of operators application leads to the CP of Figure 1.

- $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, cp_{b0})$,
- $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP_0, cp_{b1})$, $CP_1 \in \text{ISeqF}$
- $CP_2 = \otimes_{(\gg, s^2_{CP})}(CP_1, cp_{b2})$, $CP_2 \in \text{ISeqF}$
- $CP_3 = \otimes_{(\gg, s^3_{CP})}(CP_2, cp_{b3})$, $CP_3 \in \text{ISeqF}$
- $CP_4 = \otimes_{(+, s^4_{CP})}(CP_3, \{cp_{b4}, cp_{b5}\})$, $CP_4 \in \text{ISeqF} \wedge CP_4 \in \text{DC} \wedge CP_4 \in \text{PCF}$
- $CP_5 = \otimes_{(\gg, s^5_{CP})}(CP_4, cp_{b6})$, $CP_5 \in \text{ISeqF}$
- $CP_6 = \otimes_{(\gg, s^6_{CP})}(CP_5, cp_{b7})$, $CP_6 \in \text{ISeqF}$
- $CP_7 = \otimes_{(\gg, s^7_{CP})}(CP_6, cp_{b8})$, $CP_7 \in \text{ISeqF}$
- $CP_8 = \otimes_{(\circlearrowleft, s^9_{CP})}(CP_7, cp_{b9})$, $CP_8 \in \text{ISeqF}$

At the end, CP_8 is identical to the CP in Figure 1.

This example has been formalized in Event-B as an instance of the generic model introduced previously. Listing ?? extends the generic context of Listing 1 by describing set of instances used as witnesses in the generic model. It defines set of peers (see *axm*1), set of messages (see *axm*2), set of states (see *axm*3 and *axm*4), and basic CPs (see *axm*5). The obtained context is used as instances of the generic model using the RODIN platform, and the ProB[8] animator is used to check this instantiation.

Context
Constants $s0, s1, s2, s3, \ldots, Order, Deliver, Delivery\_Conf, \ldots$
Axioms
$axm1$: $partition(PEERS, \{A\}, \{L\}, \{B\})$
$axm2$: $partition(MESSAGES, \{Order\}, \{Deliver\}, \ldots)$
$axm3$: $partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \{s5\}, \ldots)$
$axm4$: $partition(A\_STATES, \{s0\_A\}, \{s1\_A\}, \ldots)$
$axm5$: $CPs\_B = \{s0 \mapsto B \mapsto Order \mapsto A \mapsto s1 \mapsto 1, s1 \mapsto A \mapsto Deliver \mapsto L \mapsto s2 \mapsto 2, CP_1, \ldots\}$
End

Listing 18: Simple procurement process within a virtual company instance

6.2 | A nonrealizable CP with choreography repair

Description. According to Basu et al,[14] the following case study describes the choreography, depicted in Figure 3, of a "simple file transfer protocol where P_1 is a client asking for the file transfer, P_2 is a file server, and P_3 initializes the communication between client and server. This CP is depicted in Figure 3. First, the client sends a message (*init*) to the server to request the server to start the transfer (*ms*). When the transfer is finished, the server sends the "transfer-finished" (*mf*) message, and the protocol terminates. However, the client may decide to cancel the transfer before hearing back from the server by sending a "cancel-finished" message (*mc*) in which case the server responds with "transfer-finished" (*mf*) message, which, again, terminates the protocol."

When projected, the CP of Figure 3 produces three peers (see Figure 4).

The choreography specification of Figure 3 is not realizable. Let us illustrate this by defining a sequence of operators. We obtain the following steps.

- $CP = \varnothing$

Basic transitions

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1$, $CP_{b1} = s_1 \xrightarrow{ms^{P1\to P2}} s_2$, $CP_{b2} = s_2 \xrightarrow{mc^{P1\to P2}} s_3$, $CP_{b3} = s_3 \xrightarrow{mf^{P2\to P1}} s_4$, $CP_{b4} = s_2 \xrightarrow{mf^{P2\to P1}} s_5$.

Composition

The two first operators, sequence and choice, are sufficient to detect violations of the sufficient conditions.

- $CP_0 = \otimes_{(\gg, s^0_{CP})}(CP, CP_{b0})$✓ $CP_1 \in$ ISeqF.
- $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP_0, CP_{b1})$ ×, $CP_2 \notin$ ISeqF.
- $CP_2 = \otimes_{(+, s^2_{CP})}(CP_1, \{CP_{b2}, CP_{b3}\})$ ×, $CP_3 \notin$ PCF

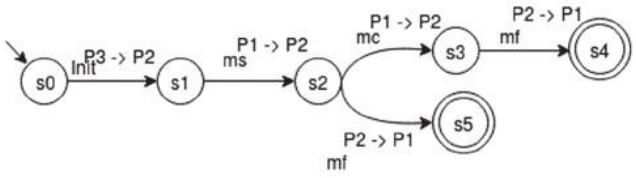

FIGURE 3 Simple file transfer protocol choreography

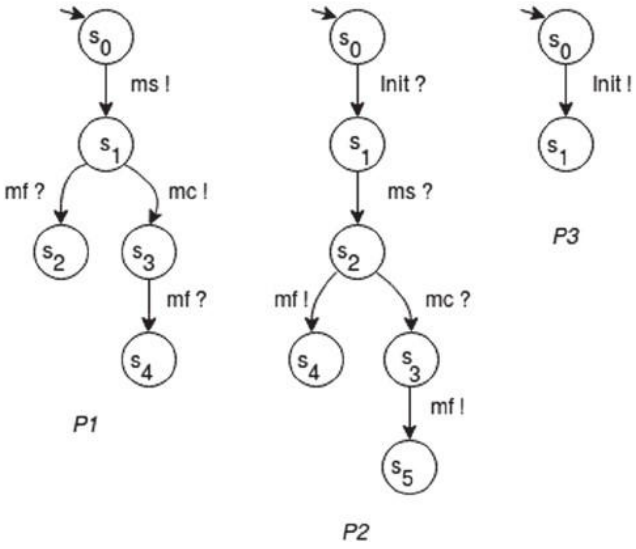

FIGURE 4 Projected peers of a simple file transfer protocol choreography

We observe that the CP_1 and CP_2 corresponding to sequence and choice operators do not satisfy the corresponding sufficient conditions. Listing 19 shows the Event-B context defined to instantiate the generic model for this case study.

Context
Constants
Axioms
$axm1: partition(PEERS, \{P1\}, \{P2\}, \{P3\}, \dots)$
$axm2: partition(MESSAGES, \{Init\}, \{ms\}, \{mc\}, \dots)$
$axm3: partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \dots)$
$axm4: partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \dots)$
$axm5: CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto mc \mapsto P2 \mapsto s3 \mapsto 3, \dots\}$
End

Listing 19: Simple file transfer protocol choreography

We observe that the sufficient conditions are not satisfied by the CP in Figure 3 both by sequence and choice operators. The instantiation of the generic model using such CP confirms the violation of sufficient conditions two times, at both the sequence and choice composition steps.

Thanks to the instantiation provided by the formal Event-B model we have built, it is possible to automatically detect the sufficient conditions violations and therefore propose a recovery or a reparation allowing to re-establish correct communications. The reparation is based on the introduction of synchronization messages exchanges. In this case, reparation shall restore the ISeqF- and PCF-violated properties. Two reparation cases can be distinguished for both sequence and branch operators as follows.

- Sequence property repair. Two possible reparations are identified in Figures 5 and 6. Following the ISeqF definition, we introduce a synchronization transition with *Sync*0 as exchanged message (highlighted with dashed line in Figures 5 and 6) establishing the ISeqF condition.
- Branch properties repair. Similarly to sequence reparation, two reparation scenarios are possible according to the PCF definition. They are depicted in Figures 7 and 8. The reparation of the choice transition requires the introduction of a synchronization message exchange *Sync*1

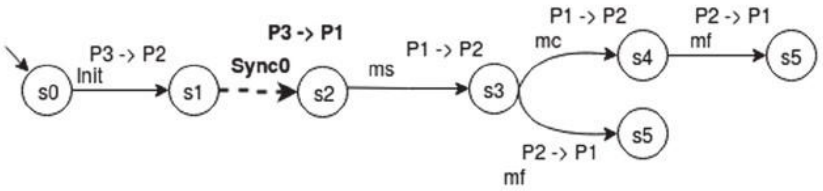

FIGURE 5 ISeqF repair proposition 5

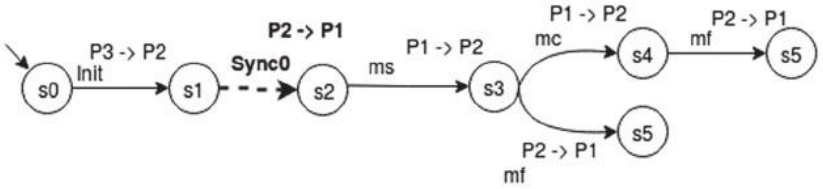

FIGURE 6 ISeqF repair proposition 6

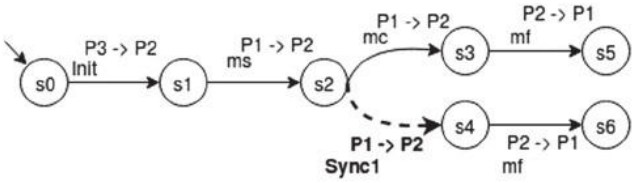

FIGURE 7 Parallel-choice freeness (PCF) repair proposition 7

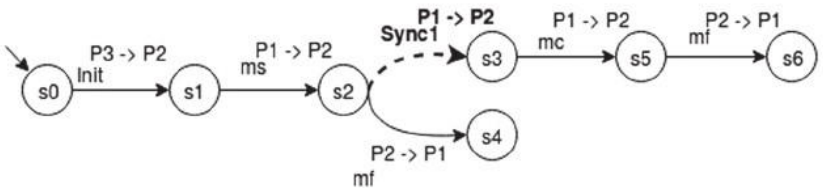

FIGURE 8 Parallel-choice freeness (PCF) repair proposition 8

(highlighted with dashed line in Figures 7 and 8) preceding one of the branches transitions. This transition exchanges a synchronization message between the same sender peer as the other branches and the receiver one.

The two proposed reparations give four reparation scenarios. Among these scenarios, we select the CP of Figure 9 to illustrate the correctness of the reparation. Concretely, the CP defined in Figure 3 is changed to the one in Figure 9, by adding the new synchronization transitions (highlighted with dashed lines). The obtained CP is realizable.

The projection of the repaired CP produces the peers of Figure 10.

Below, we show the sequence of operations allowing to build the repaired CP of Figure 9.

- $CP{=}\varnothing$

Basic transitions

$CP_{b0} = s_0 \xrightarrow{Init^{P3\to P2}} s_1$, $CP_{b1} = s_1 \xrightarrow{Sync0^{P3\to P1}} s_2$, $CP_{b2} = s_2 \xrightarrow{ms^{P1\to P2}} s_3$, $CP_{b3} = s_3 \xrightarrow{mc^{P1\to P2}} s_4$, $CP_{b4} = s_3 \xrightarrow{Sync1^{P1\to P2}} s_5$, $CP_{b5} = s_4 \xrightarrow{mf^{P2\to P1}} s_6$, $CP_{b6} = s_5 \xrightarrow{mf^{P2\to P1}} s_7$.

Composition

- $CP_1 = \otimes_{(\gg, s^1_{CP})}(CP, CP_{b0})$, ✓ $CP_1 \in \text{ISeqF}$
- $CP_2 = \otimes_{(\gg, s^2_{CP})}(CP_1, CP_{b1})$, ✓ $CP_2 \in \text{ISeqF}$
- $CP_3 = \otimes_{(+, s^3_{CP})}(CP_1, \{CP_{b3}, CP_{b5}\})$, ✓ $CP_3 \in \text{ISeqF} \wedge CP_3 \in \text{DC} \wedge CP_3 \in \text{PCF}$
- $CP_4 = \otimes_{(\gg, s^4_{CP})}(CP_3, CP_{b4})$, ✓ $CP_2 \in \text{ISeqF}$
- $CP_5 = \otimes_{(\gg, s^5_{CP})}(CP_4, CP_{b6})$, ✓ $CP_5 \in \text{ISeqF}$

Listing 20 describes the Event-B context defining the instances of the generic model (see Listing 1) defining the CP of Figure 9.

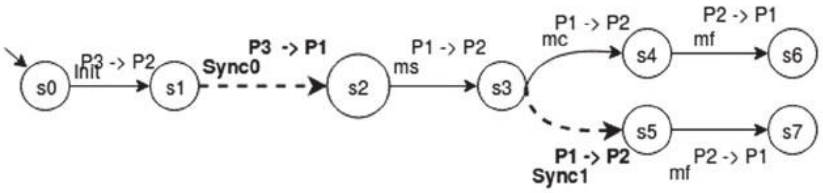

FIGURE 9 Repaired conversation protocol (CP) of a simple file transfer protocol

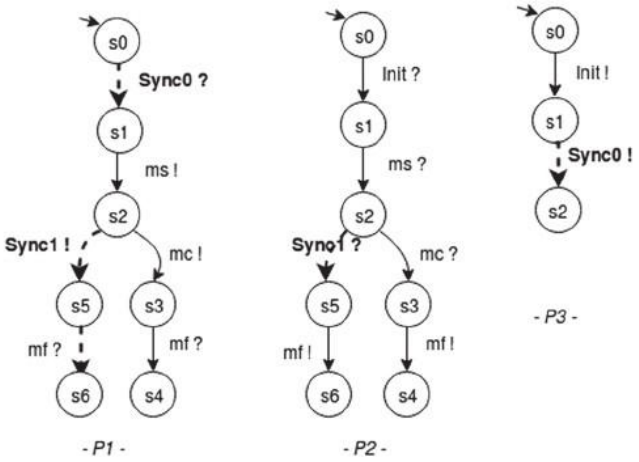

FIGURE 10 Projected peers of repaired conversation protocol (CP) of a simple file transfer protocol

Context
Constants $s0, s1, s2, s3, s4, s5, Init, Sync0, ms, mf, \dots$
Axioms
$axm1 : partition(PEERS, \{P1\}, \{P2\}, \{P3\}, \dots)$
$axm2 : partition(MESSAGES, \{Init\}, \{Sync0\}, \{ms\}, \{mf\}, \dots)$
$axm3 : partition(CP\_STATES, \{s0\}, \{s1\}, \{s2\}, \{s3\}, \{s4\}, \dots)$
$axm4 : partition(A\_STATES, \{s0\_P1\}, \{s1\_P1\}, \dots)$
$axm5 : CPs\_B = \{s0 \mapsto P3 \mapsto Init \mapsto P2 \mapsto s1 \mapsto 1, s1 \mapsto P3 \mapsto Sync0 \mapsto P1 \mapsto s2 \mapsto 2, s2 \mapsto P1 \mapsto ms \mapsto P2 \mapsto s2 \mapsto 3, \dots\}$
End

Listing 20: Repaired choreography of a simple file transfer protocol instantiation

6.3 | Other case studies

The remainder of this section describes the other benchmarks we have used to validate our approach. For each CP, we present its informal description, the associated LTS, and discuss realizability issues when our approach is applied.

- CS1: Access to web application is a real-world example depicted in Figure 11 and borrowed from Benyagoub et al.[15] It describes the access protocol to a web application. It involves three peers: a client (*cl*), a web interface (*int*), and a software application (*appli*). The CP 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*). This CP is realizable. This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.
- CS2: Database access through web application is an extension of CS1. The CP depicted in Figure 12 is borrowed from Güdemann et al.[16] It involves an additional peer with a database (*db*). The CP introduces a connection between interface and application (*Setup*), an access by the client to the database, and a log by the application. The described CP is not realizable, but a reparation is given in Güdemann et al.[16] This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.
- CS3: Unrealizable online shopping addresses an online shopping application. Following case studies are developed in Event-B, available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.
  - Unrealizable online shopping choreography. It is borrowed from Preda.[17] This application starts asking price for goods from the seller. The buyer communicates via *PriceReq* message with the seller. The seller computes the price of a product via offer message and sends the price to the buyer. If the offer is accepted, the seller sends the payment details *PayReq* to the bank. Then, the buyer authorizes the payment via the pay message. Then, either the payment successfully terminates, and the application ends with the bank acknowledging the payment to the seller and the buyer, or the payment is aborted.

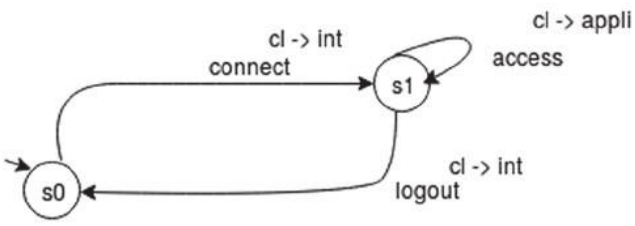

FIGURE 11 Access to a web application

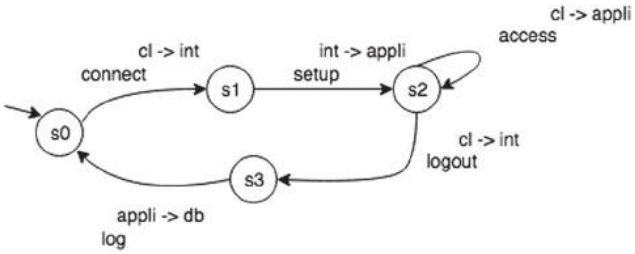

FIGURE 12 Access to a database through a web application

— Repaired online shopping. The choreography of Figure 13 is repaired using our reparation strategy to restore the set of sufficient conditions. The proposed reparation is given in Figure 14.

- CS4: Unrealizable file transfer protocol presents a nonrealizable file transfer protocol choreography borrowed from Basu and Bultan.[14] This case is described in Section 6.2, and a reparation is also proposed. This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.
- CS5: Australian working visa describes a process for an Australian working visa application. It is borrowed from Ryu et al.[18]
  Figure 15 shows a "graphical representation of a protocol for an Australian working visa application service. The visa application service is initially in the start state s_0, and service usage begins when a client sends a checkEligibility message, and the service moves to state s_1. In general, clients seeking to work in Australia can proceed to state s_6 by filling in the application, submitting their work experience, and testing their English ability, while clients reapplying for the working visa after visa expiry can go to the same state only by filling in the application and providing an employer reference letter. Overseas students who complete eligible studies in Australia proceed to state s_3 by filling in the application for overseas students and submitting graduation certificate and passport. Then, they check their application status and complete the application ending at states s_9 or s_{10}." This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.

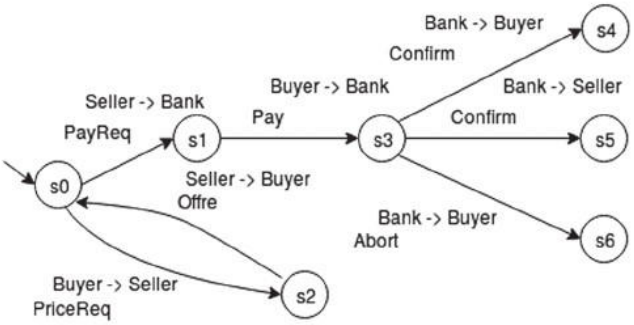

FIGURE 13 Unrealizable choreography of an online shopping

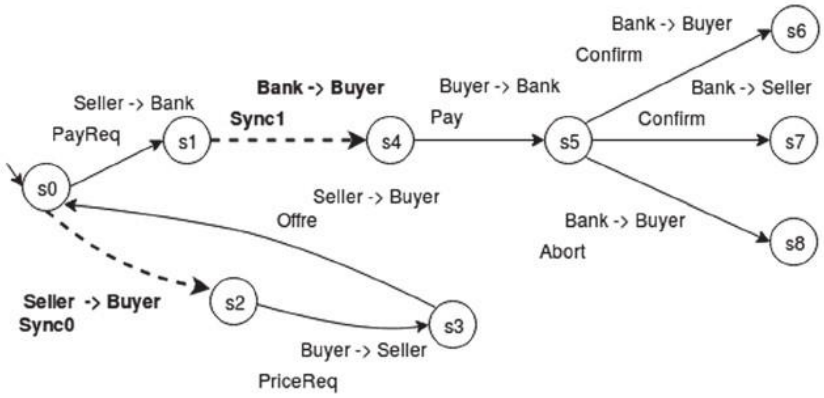

FIGURE 14 Choreography of an online shopping

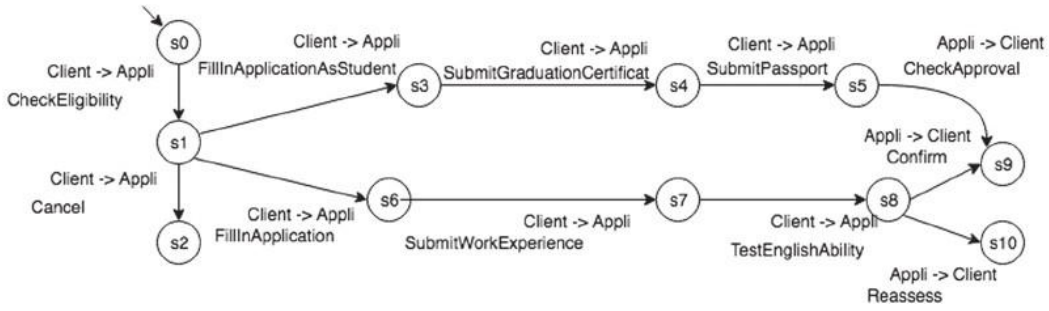

FIGURE 15 Protocol for Australian working visa

- CS6: Protocol of a fictive game is a protocol of a fictive game borrowed from Lange et al[19] and depicted in Figure 16. It consists of four peers where
  1. Alice (A) sends either *bwin* to Bob (B) or *cwin* to Carol (C) to decide who wins the game. In the former case, A fires the transition *AB!bwin* whereby the message *bwin* is put in the FIFO buffer AB from A to B, and likewise in the latter case.
  2. If B wins (that is the message *bwin* is on top of the queue AB and B consumes it by taking the transition *AB?bwin*), then he sends a notification *(close)* to C to notify her that she has lost. Symmetrically, C notifies B of her victory *(blose)*.
  3. During the game, C notifies Dave (D) that she is busy.
  4. After B and C have been notified of the outcome of the game, B sends a signal (sig) to A, while C sends a message (msg) to A.
  5. Once the result is sent, A notifies D that C is now free, and a new round starts.
  This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.
- CS7: Virtual enterprise describes a virtual enterprise CP described by Figure 1 and studied in Section 6.1. It is borrowed from Rinderle et al[13] and defines a realizable protocol. This case study is developed in Event-B. It is available at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.

7 | ASSESSMENT

In this paper, we have presented an Event-B development composed of two parts. The first one concerns the definition of correct-by-construction conversation protocols composition operators, which preserve realizability. The second one deals with a set of case studies, borrowed from the literature, which have been proved to be realizable and/or have been repaired in order to restore realizability. Below, we provide an assessment related to each of these parts.

7.1 | Event-B modeling

Table 3 gives the results of our experiments. One can observe that all the POs have been proved. A large amount of these POs have been proved automatically using the different provers associated with the RODIN platform. Interactive proofs of POs required to combine some interactive deduction rules and the automatic provers of RODIN. Few steps were required in most of the cases, and a maximum of 10 steps was reached. The most complex proofs concerned event simulation, in particular for the asynchronous level, where gluing invariants were required in the witnesses.

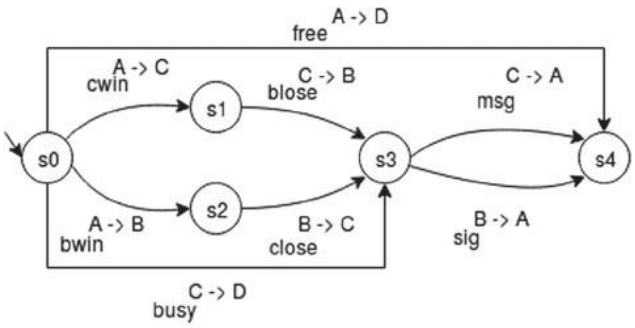

FIGURE 16 Protocol of a fictive game

TABLE 3 RODIN proofs statistics

Event-B Model	Interactive Proofs	Automatic Proofs	Proof Obligations
Abstract model	36 (64.28%)	20 (35.72%)	56 (100%)
First refinement: synchronous model	41 (40.19%)	61 (59.81%)	102 (100%)
Second refinement: asynchronous model	74 (38.75%)	117 (61.25%)	191 (100%)
Total	151 (100%)	198 (100%)	349 (100%)

7.2 | Experiments on the case studies

More generally, we discuss here our experiments done on related works examples. Since the whole development has been proved for the proposed operators, checking realizability of a given CP is reduced to checking that (a) the CP is built by composition of the defined operators and (b) that sufficient conditions hold at each application of an operator. Therefore, the instantiation for the different case studies consists in defining instances for the abstract level only.

To validate our approach and show the efficiency and the correctness of our Event-B model, we have chosen to use benchmarks issued from the literature. Table 4 summarizes the results of different case studies presented above. It gives quantitative evaluation of different criteria. For each case study, this table shows the numbers of states, peers, exchanged messages, and exchanged transitions. It states whether the CP is realizable or not and gives details of the operators used to build the CP. It also shows which properties are violated in case of nonrealizable CP. All case studies include sequences, choices, self, or cycle loops. We also have addressed the case of parallel operator by interpreting this operator as interleaving using the sequence and choice operators.

To give an idea of the number of built states during instantiation using the ProB animator, we consider the CS7—virtual enterprise case study of Figure 1. As shown in Table 4, for a CP composed of 10 states and 10 transitions, up to 298 states and 322 transitions are generated by ProB on the RODIN platform. They are used to check the correctness of instantiation (witnesses).

The whole Event-B model and all Event-B instantiations corresponding to the presented case studies are available online at http://yamine.perso.enseeiht.fr/EventB_Realisability_Models.pdf.

7.3 | A scalable approach to build correct realizable CPs: Proofs vs model checking

The developments we have conducted are all handled by the RODIN platform, which supports a proof system together with a model checker, namely, ProB.[8] Both techniques have been set up in this work.

First, realizability has been proved once for all. Indeed, the development sketched in Section 5 is generic ones, and the events are parameterized using the *ANY*-generalized substitution. Proceeding this way avoids to replay the proof and provides with a scalable approach for building realizable *CP*. Moreover, it is not required to build the composed synchronous and asynchronous projected peers compositions. The provided theorems of Section 4.2.2 define conditions that are checked only once on the constructed *CP*. Moreover, they are checked on the constructs through syntactic properties whose definitions entail scalability due to the structural nature.

Once the proofs are performed, it is sufficient to instantiate the model in order to build realizable *CP*. Therefore, they can be instantiated using any witness that fulfills the conditions to trigger this event. However, an additional proof activity was also required. It concerns consistence of the many axioms defined in this development (noncontradictory axioms). To establish axiom consistence, we have used model animation, using the ProB model checker, by providing instances for the defined sets, relations, and functions showing that the axioms are correctly inhabited, ie, model existence.

Second, establishing realizability for the particular conversation protocols corresponding to the case studies of Section 6 has been performed again using the ProB model checker. The approach consists in interpreting the deferred sets of the generic development with specific values for states, messages, basic conversation protocols, etc, and then triggering the events in order to build the final conversation protocol. Observe that the *ANY*- generalized substitution defines an existential proof obligation for which a proof can be produced by showing witnesses. ProB has been

TABLE 4 Related works: Technical report

References	States	Peers	Messages	Trans	Realizable/ Unrealizable	Operators	Properties Violated	Automatic Checking	Total States	Total Trans
CS1. Access to web application[15]	2	3	3	3	Realizable	$\gg, \circlearrowleft$	-	9 (34%)	24	23
CS2. Database access through web application[16]	4	4	5	5	Unrealizable	$\gg, \circlearrowleft$	ISeqF	7 (28%)	22	21
CS3. Unrealizable online shopping[17]	7	3	6	7	Unrealizable	$\gg, +, \circlearrowleft$	PCF,ISeqF	8 (30%)	24	23
CS3. Repaired online shopping[17]	9	3	8	9	Realizable	$\gg, +, \circlearrowleft$	-	21 (100%)	21	20
CS4. Unrealizable file transfer protocol[14]	5	2	3	4	Unrealizable	$\gg$,+	PCF	90 (63%)	98	32
CS4. Repaired file transfer protocol[14]	5	2	3	5	Realizable	$\gg$,+	-	50 (33%)	29	22
CS5. Australian working visa[18]	20	2	21	21	Realizable	$\gg, +, \circlearrowleft$	-	90 (52%)	29	32
CS6. Protocol of a fictive game[19]	5	4	8	8	Unrealizable	$\gg, +, \circlearrowleft$	PCF	6 (62%)	9	8
CS7. Virtual enterprise[13]	10	3	9	10	Realizable	$\gg, +, \circlearrowleft$	-	190 (63%)	298	322

set up to check all the properties related to the instantiation. Another alternative would consist in developing a new model refining the generic one in which witnesses are proved to be correct. This alternative may be used when model checkers do not scale up but may require interactive proofs.

8 | RELATED WORK

There exists much work on the verification of realizability, eg, Basu et al, Fu et al, Lohmann and Wolf, and Basu and Bultan.[5,20-22] Let us focus on related approaches, which propose solutions for ensuring realizability of a choreography. Carbone et al[23] identify three principles for global descriptions under which they define a sound and complete end-point projection, ie, the generation of distributed processes from the choreography description. If these rules are respected, the distributed system obtained by projection behaves exactly as specified in the choreography. The same approach is chosen for BPMN 2.0 choreographies.[24] Qiu et al[25] modify their choreography language to include new constructs (dominated choice and loop). During projection of these new operators, some communication channels are added to enforce the peers to respect the original choreography specification. However, these solutions prevent the designer from specifying what (s)he wants to and complicate the design by obliging the designer to make explicit extraconstraints in the specification, eg, by associating dominant roles with certain peers. Decker et al[26] propose a Petri net-based models for choreographies and algorithms to check realizability and local enforceability. A choreography is locally enforceable if interacting peers are able to satisfy a subset of the requirements of the choreography. To ensure this, some exchanged messages in the distributed system are disabled. Salaün et al[27] propose automated techniques to check the realizability of collaboration diagrams for different communication models. Like in our reparation strategy, in case of nonrealizability, this approach proposes to add messages directly to the peers to enforce realizability.

As compared with the above-mentioned related works, in our work, verification complexity is significantly reduced since our approach does not require the projected peers nor the recomposition of the distributed system to check realizability. Instead, we rely on sufficient conditions, defined at the CP level. There is no need to build synchronous or asynchronous peers compositions. So time and space complexities are avoided.

9 | CONCLUSION

This paper presents an a priori approach to build realizable CPs. It advocates the use of a correct-by-construction approach, which uses a set of composition operators that preserve realizability while building incrementally a given CP. The presented approach has been completely formalized in a generic Event-B development. We have used the inductive process supplied by Event-B to show that each operator with a realizable CP as an input produces another realizable CP (realizability preservation). We have defined language constructs allowing to incrementally build complex realizable CPs from a set of basic realizable ones. Moreover, we have shown that the proposed development is generic and can be instantiated to any set of basic and composite CPs, showing scalability of the approach. We also propose a novel incremental reparation technique based on our sufficient conditions.

In this paper, we have validated and experimented our approach and its formal Event-B models on benchmark case studies commonly used in the literature. The interest of this work is double. On the one hand, these benchmarks are used to validate our approach through instantiation. On the other hand, they show that the defined sufficient conditions are not restrictive and cover the variety of case studies available in the literature, in particular for the definition of business processes.

Moreover, this paper shows that the proposed approach scales to CPs of arbitrary sizes. It does not require to build synchronous or asynchronous peers compositions like in traditional a posteriori approaches.

As a short-term perspective, we plan to formalize our reparation technique using Event-B in order to support on the fly CP reparation instead of checking the whole repaired CP. A new reparation event should be added to our Event-B model.

We also aim at extending our model with a new operator to enable composition of entire CPs instead of building CPs from scratch that requires incremental composition using basic CPs. As a long-term perspective, three main research paths have been identified.

First, in addition to the sufficient conditions we defined, we will propose the definition of necessary conditions for specific conversation protocols templates. Second, in the same manner as for finite messages exchanges, studying the case of infinite messages exchanges should be studied for specific cases of conversation protocols. Finally, we aim at providing the designers with an engine for automatic instantiation of realizable CPs.

ACKNOWLEDGMENTS

The research reported in this paper has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET Center SCCH.

ORCID

Sarah Benyagoub https://orcid.org/0000-0002-6859-1092
Atif Mashkoor https://orcid.org/0000-0003-1210-5953

REFERENCES

1. Benyagoub S, Ouederni M, Aït-Ameur Y, Mashkoor A. Incremental construction of realizable choreographies. In: NASA Formal Methods—NFM 2018; 2018:1-19.
2. Benyagoub S, Ouederni M, Aït-Ameur Y, Mashkoor A. Scalable correct-by-construction conversation protocols with Event-B: validation, experiments and benchmarks. In: 23rd International Conference on Engineering of Complex Computer Systems (ICECCS); 2018:209-212.
3. Benyagoub S, Ouederni M, Aït-Ameur Y, Mashkoor A. Handling reparation in incremental construction of realizable conversation protocols. In: New Trends in Model and Data Engineering—MEDI International Workshops, DETECT, MEDI4SG, IWCFS, REMEDY; 2018:159-166.
4. Brand D, Zafiropulo P. On communicating finite-state machines. *J ACM (JACM)*. 1983:323-342.
5. Basu S, Bultan T, Ouederni M. Deciding choreography realizability. In: Proc of POPLŠ12 ACM; 2012:191-202.
6. Abrial JR. Modeling in Event-B. *System and Software Engineering, Cambridge, Ed.* 2010.
7. Abrial JR, Butler M, Hallerstede S, Hoang TS, Mehta F, Voisin L. RODIN: an open toolset for modelling and reasoning in Event-B. *Int J Softw Tools Technol Transfer*. 2010;12(6):447-466.
8. Leuschel M, Butler MJ. ProB: a model checker for B,; 2003:855-874.
9. Hopcroft JE, Ullman JD. *Introduction to Automata Theory, Languages and Computation*: Addison Wesley; 1979.
10. Farah Z, Aït-Ameur Y, Ouederni M, Tari K. A correct-by-construction model for asynchronously communicating systems. *Int J Softw Tools Technol Transfer*. 2017;19(4):465-485.
11. Finkel A, Lozes É. Synchronizability of communicating finite state machines is not decidable. In: 44th International Colloquium on Automata, Languages, and Programming, ICALP; 2017:122:1-122:14.
12. Abadi M, Lamport L. The existence of refinement mappings. *Theor Comput Sci*. 1991:253-284.
13. Rinderle S, Wombacher A, Reichert M. On the controlled evolution of process choreographies. In: Proc. of ICDE; 2006:100-124.
14. Basu S, Bultan T. Automated choreography repair. In: Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. 9633 of Lecture Notes in Computer Science Stevens P, Wasowski A, eds. Springer; 2016:13-30.
15. Benyagoub S, Ouederni M, Aït-Ameur Y. Towards correct evolution of conversation protocols. In: Proc of VECOSŠ16; 2016:193-201.
16. Güdemann M, Salaün G, Ouederni M. Counterexample guided synthesis of monitors for realizability enforcement. In: Proc. of ATVAŠ12; 2012:238-253.
17. Preda MD, Gabbrielli M, Giallorenzo S, Lanese I, Mauro J. Dynamic choreographies—safe runtime updates of distributed applications. In: Proc of COORDINATIONŠ15; 2015:67-82.
18. Ryu SH, Casati F, Skogsrud H, Benatallah B, Saint-Paul R. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Trans Web (TWEB)*. 2008;2(2):13.
19. Lange J, Tuosto E, Yoshida N. From communicating machines to graphical choreographies. In: Proceedings of the 42nd Annual ACM POPL; 2015:221-232.
20. Fu X, Bultan T, Su J. Conversation protocols: a formalism for specification and verification of reactive electronic services. *Theoretical Computer Science*. 2004:19-37.
21. Lohmann N, Wolf K. Realizability is controllability. In: International Workshop WSFM; 2009:110Ű-127.
22. Basu S, Bultan T. On deciding synchronizability for asynchronously communicating systems. *Theor Comput Sci*. 2016;656:60-75.
23. Carbone M, Honda K, Yoshida N. Structured communication-centred programming for web services. In: Proc of ESOPŠ07; 2007.
24. Rosing M, White S, Cummins F, Man dH. Business process model and notation—BPMN. *The Complete Business Process Handbook: Body of Knowledge From Process Modeling to BPM, Volume I*; 2015:429-453.
25. Qiu Z, Zhao X, Cai C, Yang H. Towards the theoretical foundation of choreographyProceedings of the 16th international conference on World Wide Web. ACM; 2007:973-982.
26. Decker G, Weske M. Local enforceability in interaction Petri nets. In: International Conference on Business Process Management. Springer; 2007:305-319.
27. Salaün G, Bultan T. Realizability of choreographies using process algebra encodings. In: International Conference on Integrated Formal Methods; 2009:167-182.

How to cite this article: Benyagoub S, Aït-Ameur Y, Ouederni M, Mashkoor A, Medeghri A. Formal design of scalable conversation protocols using Event-B: Validation, experiments, and benchmarks. *J Softw Evol Proc*. 2020;32:e2209. https://doi.org/10.1002/smr.2209